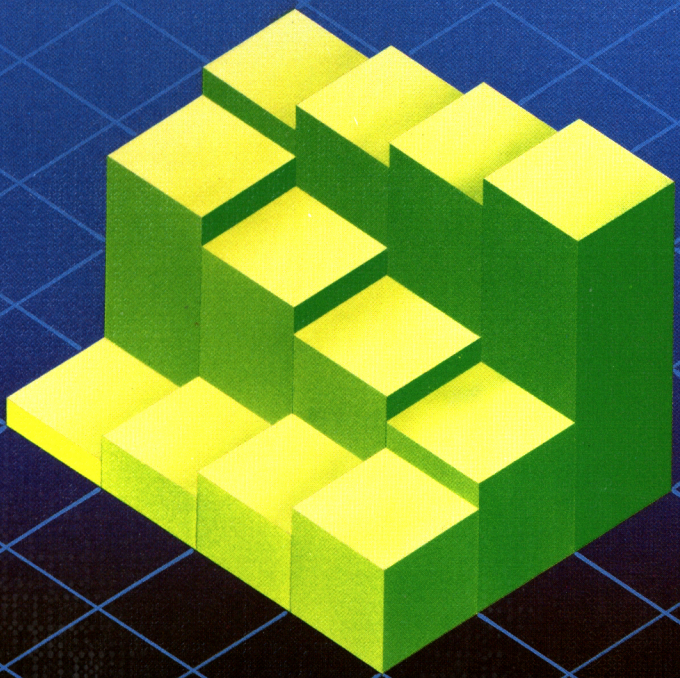
Manfred Walter Thoma

CPC-464/664 Praxis

Band 1: Schwerpunkt Grafik



Hüthig

Manfred Walter Thoma · CPC 464/664 - Praxis

Manfred Walter Thoma

CPC-464/664 – Praxis

Band 1: Schwerpunkt Grafik

Dr. Alfred Hüthig Verlag Heidelberg

Diejenigen Bezeichnungen von im Buch genannten Erzeugnissen, die zugleich eingetragene Warenzeichen sind, wurden nicht besonders kenntlich gemacht. Es kann also aus dem Fehlen der Markierung ® nicht geschlossen werden, daß die Bezeichnung ein freier Warenname ist. Ebensowenig ist zu entnehmen, ob Patente oder Gebrauchsmusterschutz vorliegen.

CIP-Kurztitelaufnahme der Deutschen Bibliothek

Thoma, Manfred Walter:
[CPC-vierhundertvierundsechzig-Praxis]
CPC-464-Praxis / Manfred Walter Thoma. —
Heidelberg : Hüthig

Bd. 1. Schwerpunkt Grafik. — 1985. —
ISBN 3-7785-1149-1

© 1985 Dr. Alfred Hüthig Verlag GmbH, Heidelberg
Printed in Germany
Satz und Druck: Druckerei Bitsch GmbH, Birkenau

Inhaltsverzeichnis

0.0 Vorwort 11

1.0 Einleitung 13

2.0 Grafik 17
2.1 27 Farben 18
2.2 PEN und PAPER 19
2.3 INK - neue Farben in die Fässer 20
2.4 Farbe des BORDERs 21
2.5 Blinkgeschwindigkeit - SPEED INK 22
2.6 Transparent-Modus 23
2.7 Inverse-Modus 24

3.0 Bildschirmspeicher 25
3.1 Organisation des Bildschirmspeichers 25
3.2 Farbe und Form 28
3.3 'Garfield' in Multicolour 31

4.0 Der Zeichensatz 35
4.1 Das Zeichen-ROM 36
4.2 SYMBOL AFTER - Zeichensatz kopieren 37
4.3 Zeichen auslesen 40
4.4 SYMBOL - neue Zeichen erstellen 41

5.0 WINDOW - das Fenster im Bildschirm 51

6.0 Hochauflösende Grafik 57
6.1 ORIGIN - der Nullpunkt 57
6.2 PLOT - der Punkt 59
6.3 Die Ellipse 61
6.4 Der Grafik-Cursor 63
6.5 MOVE - Positionieren des Grafik-Cursors 63
6.6 DRAW - Ziehen einer Linie 64
6.7 Rechtecke 65
6.8 Quader - ein Körper 68

7.0 Grafische Darstellung von Werten 73
7.1 Balkendiagramm 73
7.2 Werte-Plot 77
7.3 Tortendiagramm 79
7.4 Beschriften von Grafiken - TAG/TAGOFF 82
7.5 3D-Balkendiagramm 83

8.0 Das Grafikfenster 91

9.0 Funktions-Plot 95

10.0 3D-Funktions-Plot 101
10.1 Sichtbar oder unsichtbar? 102

11.0 TEST und TESTR 109

12.0 Die Centronics-Schnittstelle 111
12.1 Das Kabel zum Drucker 111
12.2 PRINTen auf dem Drucker 112
12.3 Die Hardcopy 113
12.4 Grafikhardcopy 113
12.5 Texthardcopy 119

13.0 Der Joystick 123
13.1 JOY - Abfrage des Joysticks 124

14.0 ELOCAD - ein Zeichenprogramm 127

15.0 Tastatur 137
15.1 KEY DEF - Neubelegung der Tasten 138
15.2 Erweiterungszeichen 139
15.3 KEY-Belegen der Funktionstasten 140
15.4 Programmierhilfe 141
15.5 Auslesen der Funktionstastenbelegung 143

16.0 Die Kontroll- und Steuerzeichen 145

17.0 Fehlerbehandlung 151

18.0 Interrupt - Unterbrechungen 155
18.1 AFTER - einmaliger Aufruf 156
18.2 EVERY - ständiger Aufruf 158
18.3 REMAIN - Timer abschalten 163
18.4 DI und EI - zeitweiliges Ausschalten 164

19.0 Externe Datenspeicher 167
19.1 Kassette und Diskette 167
19.2 SAVE - Sichern von Programmen 168
19.3 LOAD - Laden von Programmen 170
19.4 MERGE - Programme zuladen 170
19.5 RUN „...“ - Laden und Starten 171
19.6 CAT - Inhaltsverzeichnis 171

20.0 Dezimalsystem 173
20.1 Binärsystem 173
20.2 Logische Arithmetik 176
20.3 Zerlegen in High- und Lowbyte 179
20.4 Hexdezimalsystem 181

21.0 MEMORY - ein Spiel 183

Register 187

Programmverzeichnis

01 Farbdemonstration ..... Seite 20
02 Speed-Ink ..... Seite 22
03 Transparentdemonstration ..... Seite 23
04 Inversedemonstration ..... Seite 24
05 Garfield ..... Seite 31
06 Beispiele ..... Seite 35
07 Zeichen auslesen ..... Seite 41
08 BTX-Zeichen ..... Seite 42
09 Zeichengenerator ..... Seite 44
10 5 Fenster ..... Seite 53
11 Farbige Fenster ..... Seite 54
12 Ellipse ..... Seite 62
13 Rechtecke1 ..... Seite 66
14 Rechtecke2 ..... Seite 67
15 Quader ..... Seite 70
16 Balkendiagramm ..... Seite 75
17 Werte-Plot ..... Seite 77
18 Tortendiagramm ..... Seite 80
19 3D-Balkendiagramm ..... Seite 86
20 Grafikwindow ..... Seite 92
21 Funktionsplot ..... Seite 97
22 3D-Funktionsplot ..... Seite 104
23 Grafikhardcopy (BASIC) ..... Seite 115
24 Grafikhardcopy (MPRG) ..... Seite 115
25 Grafikhardcopy (Assembler) ..... Seite 116
26 Texthardcopy (BASIC) ..... Seite 120
27 Texthardcopy (MPRG) ..... Seite 121
28 Texthardcopy (Assembler) ..... Seite 121
29 Joystickdemonstration ..... Seite 125
30 ELOCAD ..... Seite 129
31 Programmierhilfe ..... Seite 141
32 KEYLIST ..... Seite 143
33 Analog-Uhr ..... Seite 159
34 Interruptdemonstration ..... Seite 163
35 Memory ..... Seite 183

0.0 Vorwort

Ein neues Gesicht hat den Homecomputermarkt tüchtig durcheinandergewirbelt und die Grenzen zwischen Home- und Personal Computern noch mehr verwischt. Die Leistungsfähigkeit des Schneider CPC-464 liegt deutlich über dem allgemeinen Homecomputerniveau und grenzt mit seinen Leistungsmerkmalen an die der Personal Computer.

Es muß nicht weiter erwähnt werden, daß der Schneider CPC-464 heute eindeutig der Leistungsfähigste seiner Klasse ist. Der Preis von knapp 900,— DM incl. Monitor und Datenrecoder macht ihn nur noch attraktiver. Er hat in fast allen Bereichen einen neuen Standard gesetzt, an denen zukünftig alle anderen Computer der unteren Preisklasse zu messen sind. Hierzu zählen vor allem:

* 80 Zeichen pro Zeile
* 128000 Bildpunkte
* Window-Technik
* Leistungsstarker BASIC-Interpreter
* Interruptsteuerung
* CP/M fähig

Auf Grund der Fülle und Leistungsvielfalt war es nicht möglich, den Schneider CPC-464 in einem Buch so zu behandeln, daß alle Möglichkeiten in ausreichendem Maße behandelt werden konnten.

Der Ihnen vorliegende 1. Band beschäftigt sich daher schwerpunktmäßig mit der Grafik. Es ist besonders darauf geachtet worden, daß alle behandelten Probleme und Programmiertechniken immer einen praktischen Bezug haben. Alle Programme sind aus einem konkreten Bereich entwickelt und mit den behandelten Befehlen realisiert worden. Neben unzähligen Beispielen finden Sie viele Programme und Unterprogramme, die Ihnen die zukünftige Arbeit mit dem Schneider CPC-464 erheblich erleichtern. So finden Sie Programme von der Multicolorgrafik bis hin zum kompletten „CAD"-System, vom Tortendiagramm bis zum 3D-Funktionsplot, von der 3-dimensionalen Wertdarstellung bis zur Analoguhr. Alle Programme sind so ausgelegt, daß sie ohne große Schwierigkeiten Ihren ganz speziellen Wünschen angepaßt werden können. Und doch handelt es sich bei diesem Buch nicht um eine Programmsammlung im eigentlichen Sinne. Es ist ein Arbeits- und Handbuch, welches durch praktische Anwendungen (Programme) ergänzt ist, die den professionellen Charakter des Schneider CPC-464 deutlich unterstreichen.

Neben der Grafik ist weiterhin die Centronics-Schnittstelle mit mehreren Hardcopy-Programmen, der Joystick, die Tastatur und die Interruptsteuerung ausführlich behandelt. Die Kassetten- und Diskettenbedienung findet ebenso ihren Platz wie alle Besonderheiten des BASIC-Interpreters.

Im Band 2 zum Schneider CPC-464 liegt der Schwerpunkt bei der Datenverwaltung (Diskette/Kassette), und dem CP/M-Betriebssystem.

Die Datenverwaltung beschäftigt sich mit allen Möglichkeiten der Verwaltung und Organisation von Daten auf Diskette und Kassette. Dazu gehören neben Sortier- und Suchroutinen natürlich auch wieder viele komplette Anwenderprogramme bis hin zum „Datenbanksystem". Durch die CP/M (Control Program for Microcomputer) Fähigkeit des CPC-464 eröffnet sich eine neue Welt für den Programmierer. Daher beschäftigt sich ein Teil von Band 2 ausführlich mit dem CP/M Betriebssystem. Neben Erläuterungen zur Funktion, Bedienung und Anwendung der zur Diskette mitgelieferten CP/M-Hilfsprogramme, finden Sie einen Einstieg in die Programmierung unter CP/M.

Hamburg, April 1985

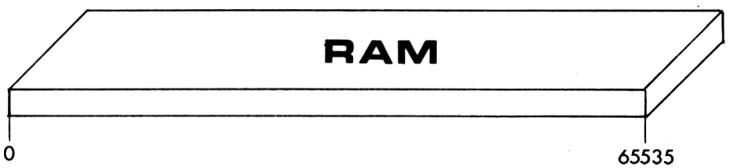
1.0 Einleitung

Die technische Entwicklung, gerade im Bereich der Elektronik, hat immer neue und vor allem leistungsfähigere Computer geschaffen. Ein typischer Vertreter der neuen (Home-) Computergeneration ist der Schneider CPC-464* (CPC = Colour Personal Computer). Mit nur 25 IC's ist ein Computer entstanden, der vor Jahren noch einen Schrank voll Elektronik benötigt hätte.

Das Herz des CPC-464 bildet der Mikroprozessor Z80. Der Z80 wurde von der Firma Zilog entwickelt und ist ein Nachfolger des 8080 Mikroprozessors (ebenfalls Zilog). Der Z80 unterscheidet sich vom 8080 vor allem durch einen erweiterten Befehlssatz und erheblich mehr Register (interne Speicher des Prozessors). Heute ist der Z80 der verbreiteste Mikroprozessor überhaupt.

Der Z80 besitzt einen 8-Bit Datenbus und einen 16-Bit Adressbus. Ein „bus" ist der Weg für elektronische Signale, um von einem Baustein zu einem anderen zu gelangen. Über den Datenbus werden Informationen, die eine Größe von einem Byte (8 Bit) besitzen, übertragen. Der Adressbus teilt dem Speicher mit, welche Speicherstelle als nächstes beschrieben oder gelesen wird. Durch die 16-Bit Struktur lassen sich 65536 Speicherstellen (2^16 = 65536 = 64 kByte) ansprechen.

Der gesamte adressierbare Speicherraum des CPC-464 ist mit einem RAM (Random Access Memory) belegt. Ein RAM ist ein Speicher für den beliebigen Zugriff, d.h. er kann gelesen und beschrieben werden (PEEK und POKE).



Nach Abschalten der Betriebsspannung verlieren RAM-Bausteine ihren Inhalt und vorhandene Daten gehen verloren. Im RAM-Bereich werden vor allem Programme, Daten und die Bildschirminformation abgelegt.

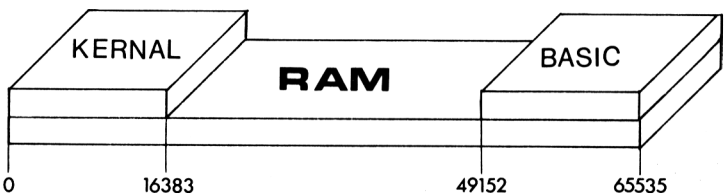
Das Betriebssystem und der BASIC-Interpreter sind die eigentlichen Leistungsträger eines Computers. Das Betriebssystem ist der Koordinator im

* Wenn nicht weiter erwähnt, beziehen sich alle Angaben ebenfalls auf den CPC-664.

CPC-464. Es steuert die Bildschirm- oder Druckerausgabe, fragt die Tastatur ab etc. Damit der Rechner unsere Sprache, in diesem Fall BASIC, versteht, übersetzt der BASIC-Interpreter Befehle in einen für den Mikroprozessor verständlichen Code. Beide, Betriebssystem und BASIC-Interpreter, sind (Maschinen-) Programme, die immer zur Verfügung stehen müssen. Aus diesem Grund können sie nicht in einem RAM-Baustein untergebracht sein. Ein spezieller Baustein kann Daten speichern, unabhängig davon, ob der Computer an- oder ausgeschaltet ist. Er nennt sich ROM (Read Only Memory = nur lese Speicher). Wie der Name schon besagt, können diese Bausteine nur ausgelesen, nicht aber beschrieben werden (PEEK). In einem 32 kByte ROM sind die beiden Programme (Betriebssystem und BASIC) untergebracht. Sicherlich werden Sie sich jetzt fragen, wo dieser 32 kByte Speicher noch untergebracht ist, wo doch der gesamte adressierbare Speicherraum mit einem RAM belegt ist? Bestimmte Speicherbereiche sind mehrfach belegt, d.h. ein weiterer Speicher liegt parallel („oben drauf") zu einem anderen. Mit Hilfe des sogenannten „Bank-Switching" (Arbeitsbereich umschalten) ist es möglich, eine bestimmte „Etage" auszuwählen. Betriebssystem und BASIC-Interpreter sind in 2 Teile a 16 kByte geteilt und liegen parallel zu den Bereichen:

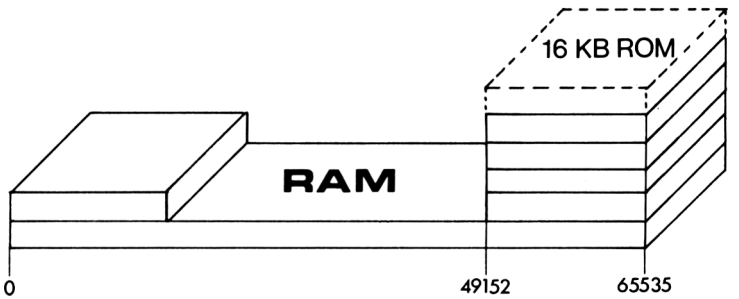
&0000 - &4000	Betriebssystem	(0-16383)
&C000 - &FFFF	BASIC-Interpreter	(49152-65535)

(Hexadezimale Zahlen sind mit einem vorgestellten „&"-Zeichen gekennzeichnet. Falls Ihnen die Bedeutung und der Umgang mit hexadezimalen Zahlen nicht geläufig ist, schlagen Sie bitte im Kapitel 20.0 nach. Dort finden Sie eine grundlegende Einführung in die verschiedenen, für die Programmierung wichtigen Zahlensysteme.)



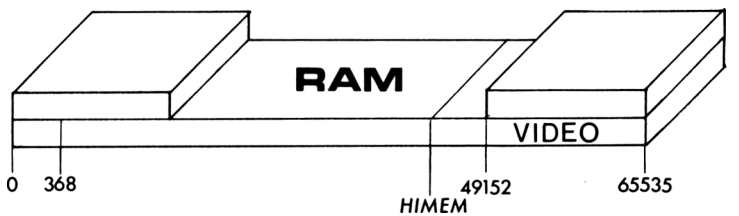
Das Umschalten zwischen den verschiedenen Speicherebenen übernimmt u.a das sogenannte „Gate Array", ein speziell für den CPC-464 entwickelter Baustein. Der Bereich &C000 bis &FFFF (49152-65535) läßt sich nicht nur dop-

pelt belegen, sondern es sind hier bis zu 252 Ebenen a 16 kByte (ROM) möglich. Damit wären bis zu 4 MByte ansprechbar. Hier liegt z.B. bei angeschlossener Diskettenstation ein weiterer ROM, in dem alle für die Diskettenstation notwendigen Routinen (DOS) abgelegt sind.



In dem 64 kByte großen RAM-Baustein befinden sich Bereiche, die für spezielle Aufgaben reserviert sind. So liegt im Bereich &C000 bis &FFFF (49152-65535) der sogenannte Bildschirmspeicher. Hier befinden sich alle Informationen, die auf dem Bildschirm sichtbar sind (siehe 3.0). Im Bereich von &0000 bis &016F (0-367) liegt u.a eine Kopie der ersten 64 Bytes des Betriebssystems (Restart-Routinen) und ein Zwischenspeicher (Puffer) für Tastatureingaben. Ab Adresse &0170 (368) beginnt der für den Programmierer frei zur Verfügung stehende Speicherraum (BASIC-Startadresse) und er endet mit Adresse &AB7F (43903) (HIMEM). Bei angeschlossener Diskettenstation DDI-1 oder beim CPC-664 verkleinert sich dieser Bereich um &0504 Bytes (1284) und endet somit bei Adresse &A67B (42619). Der frei zur Verfügung stehende Speicherraum beträgt damit bei

Kassettenbetrieb	43533 Bytes und bei
Diskettenbetrieb	42249 Bytes.



Diese kurze Einführung in den Speicheraufbau des CPC-464 soll an dieser Stelle ausreichend sein. Im weiteren Verlauf des Buches werden einige Speicherbereiche detaillierter erklärt.

2.0 Grafik

Ein wesentliches Leistungsmerkmal eines Computers sind heute seine Grafikfähigkeiten. Unter Grafik versteht man im allgemeinen die Fähigkeit, Buchstaben, Zeichen, Linien und Farben auf dem Bildschirm zu erzeugen. Wird heute von Grafikfähigkeit gesprochen, ist ausschließlich die Möglichkeit der hochauflösenden Grafik gemeint (HIRES-high resolution). Unter hochauflösender Grafik versteht man das programmierte Setzen und Löschen von einzelnen Punkten auf dem Bildschirm. Aus je mehr Punkten der Bildschirm besteht, desto feiner werden die Grafiken.

Der Bildschirm des CPC-464 besteht aus 128000 Bildschirmpunkten: 640 Punkte in der X-Achse und 200 Punkte in der Y-Achse. Im Heim- und Homecomputerbereich ist eine Auflösung von 128000 Punkten wie beim CPC-464 sehr ungewöhnlich und bisher nur bei Computern der gehobenen Preisklasse zu finden.

Insgesamt sind bis zu 16 verschiedene Farben gleichzeitig auf engstem Raum dargestellbar. Diese 16 Farben können frei aus einer Palette von 27 Farben gewählt werden.

Zwei weitere, im Homecomputerbereich nahezu unbekannte Möglichkeiten bietet der CPC-464:

* 80 Zeichen pro Zeile
* 8 Windows (Fenster)

Durch die Fähigkeit, 80 Zeichen pro Zeile darzustellen, hebt sich der CPC-464 deutlich von anderen Homecomputern ab. Die Lesbarbeit der Zeichen ist speziell beim grünen Monochrom-Bildschirm ausgezeichnet.

Da Text und Grafik auf demselben Bildschirm dargestellt werden, ist das Mischen von Text und Grafik, z.B. für das Beschriften von Diagrammen, ohne weiteres möglich.

Ebenso ist die WINDOW-Technik (eng.: window = Fenster, Aufteilen des Bildschirms in kleine, unabhängige Bildschirme) mit bis zu 8 Fenstern ein außergewöhnliches Leistungsmerkmal. Es macht wirklich Spaß, mit ihnen zu arbeiten.

Sämtliche Grafik-Funktionen des CPC-464 sind mit BASIC-Befehlen ansprechbar und bieten dem Programmierer ein sehr leistungsfähiges Werkzeug.

2.1 27 Farben

Aus einer Palette von insgesamt 27 Farben können je nach benutztem Modus 16, 4 oder 2 (verschiedene) Farben ausgewählt und gleichzeitig auf dem Bildschirm dargestellt werden:

MODUS 0 : 20 Zeichen	16 Farben (160*200 Punkte)
MODUS 1 : 40 Zeichen	4 Farben (320*200 Punkte)
MODUS 2 : 80 Zeichen	2 Farben (640*200 Punkte)

Je größer die Auflösung ist, desto geringer ist die Anzahl der gleichzeitig zu verwendenden Farben. Im MODUS 2 sind es z.B. nur noch zwei Farben.

Jede der 27 Farben ist einer Kennzahl (0 bis 26) zugeordnet. Aus der nachfolgenden Tabelle sind alle Kennzahlen mit den zugehörigen Farben zu entnehmen:

0	Schwarz	14	Pastellblau
1	Blau	15	Orange
2	Hellblau	16	Rosa
3	Rot	17	Pastellmagenta
4	Magenta	18	Hellgrün
5	Hellviolett	19	Seegrün
6	Hellrot	20	helles Blaugrün
7	Purpur	21	Limonengrün
8	helles Magenta	22	Pastellgrün
9	Grün	23	Pastellblaugrün
10	Blaugrün	24	Hellgelb
11	Himmelblau	25	Pastellgelb
12	Gelb	26	Leuchtendweiß
13	Weiß		

Aus diesen 27 Farben können Sie die maximal 16 zu benutzenden Farben auswählen. Es stehen 16 „Tintenfässer" zur Verfügung, die jeweils mit einer (bzw. zwei) Farbe gefüllt werden können. Diese „Fässer" sind nach dem Einschalten durch das Betriebssystem schon mit verschiedenen Farben gefüllt. Hierbei ist zu beachten, daß sich je nach Modus 16, 4, oder nur 2 verschiedene Farben in den „Fässern" befinden. Die „Fässer" sind von 0 bis 15 durchnummeriert und haben nach dem Einschalten (oder reset) folgenden Inhalt (Farbnummer):

Faßnummer	MODUS 0	MODUS 1	MODUS 2
00	1	1	1
01	24	24	24
02	20	20	1
03	6	6	24
04	26	1	1
05	0	24	24
06	2	20	1
07	8	6	24
08	10	1	1
09	12	24	24
10	14	20	1
11	16	6	24
12	18	1	1
13	22	24	24
14	1/16	1	1
15	16/11	6	24

Sie sehen, daß sich nach dem Einschalten des CPC-464 (MODUS 1) im Faß 0 die Farbe Blau (1) und in Faß 1 die Farbe Hellgelb (24) befindet.

2.2 PEN und PAPER

Mit den Befehlen PEN (Stift, Schreiber) und PAPER (Papier) ist die aktuelle Zeichen- und Hintergrundfarbe zu bestimmen. Nach dem Einschalten ist die Hintergrundfarbe (PAPER) mit dem Inhalt des Fasses 0 gefüllt (1=Blau) und die Zeichenfarbe (PEN) mit den Inhalt aus Faß 1 (24=Hellgelb). Um die Zeichenfarbe zu ändern, muß dem CPC-464 mitgeteilt werden, aus welchem Faß er ab sofort seine neue Zeichenfarbe holen soll. Mit

```
PEN faßnummer
PEN 2
```

ist die Zeichenfarbe nun die Farbe, die sich im Faß 2 befindet (20=helles Blaugrün). Ebenso läßt sich die Hintergrundfarbe mit dem Befehl:

```
PAPER faßnummer
PAPER 3
```

auf Hellrot einstellen (im Faß 3 befindet sich die Farbkennzahl 6). Mit

PEN 0: PAPER 1

wird die Hintergrundfarbe auf Gelb (Faß 1) und die Zeichenfarbe auf Blau (Faß 0) gelegt. Die READY-Meldung erscheint nun in blauer Schrift auf hellgelbem Hintergrund. Nach dem Löschen des Bildschirms mit CLS erscheint der gesamte Hintergrund in Hellgelb.

2.3 INK - neue Farben in die Fässer

Mit dem Befehl INK (Tinte) ist es möglich, andere als die vorgegebenen Farben in die Fässer zu legen.

INK faßnummer , farbkennzahl

Um in das Faß 0 die Farbe Rosa (16) zu legen, ist der Befehl:

INK 0, 16

zu verwenden. Alle Zeichen (oder der Hintergrund), die in der Farbe aus Faß 0 geschrieben worden sind, erscheinen sofort in der neuen Farbe.

```
1000 REM ************************************ M.W.Thoma **
1010 REM *                                              *
1020 REM *              Farbdemonstration               *
1030 REM *                                              *
1040 REM ***************************************************
1050 MODE 1
1060 PRINT " >Farbdemonstration<"
1070 LOCATE 1,3
1080 INPUT " >Welcher MODUS (0-2) :";m
1090 IF m=0 THEN z=15:GOTO 1140
1100 IF m=1 THEN z=3 :GOTO 1140
1110 IF m=2 THEN z=1 :GOTO 1140
1120 PRINT CHR$(7):'Fehlerton
1130 GOTO 1070
1140 MODE m
1150 PRINT "MODUS";m
1160 LOCATE 1,3
1170 FOR f=1 TO z
1180 PEN f:PRINT "Farbe in Fass";f
1190 NEXT
1200 PEN 1
```

```
1210 LOCATE 1,22
1220 INPUT "Fass,Farbe :";fass,farbe
1230 INK fass,farbe
1240 GOTO 1160
1250 REM ************************************************
```

Im obigen Demonstrationsprogramm werden alle Farben, die im jeweiligen Modus möglich sind, dargestellt. Sie können den Inhalt der Fässer ändern und sich das Ergebnis auf dem Bildschirm ansehen. Die Hintergrundfarbe entspricht auch in diesem Programm der Farbe aus Faß 0. Beachten Sie, daß es keinen Einfluß auf die darstellbaren Farben hat, wenn Sie in ein „nicht vorhandenes" Faß, z.B. Faß 4 im Modus 1, eine Farbe legen. Andererseits werden diese Farben aber tatsächlich in die Fässer gelegt und können in einem anderen Modus Auswirkungen haben.

Im Modus 0 sind Ihnen sicherlich die Fässer 14 und 15 aufgefallen. Ist im Modus 0 das Faß 15 als Zeichenfarbe gewählt,

MODE 0: PEN 15

erscheint die READY-Meldung blinkend in den Farben Rosa/Himmelblau (16/11). Es ist möglich, jedes Faß mit einer zweiten Farbe, in der der Schriftzug blinken soll, zu belegen.

INK faßnummer, farbe1, farbe2

Mit dem Befehl

INK 1,0,26

werden in Faß 1 die Farben Schwarz (0) und Leuchtendweiß (26) gelegt, die die Blinkfarben angeben.

2.4 Farbe des BORDERs

Unter dem „Border" (Rand, Einfassung) versteht man den Teil des Bildschirms, welcher für die Darstellung von Zeichen nicht genutzt wird. Der Grund dafür liegt in den Bildverzerrungen an den Bildschirmseiten. Der Border wird nicht gleichzeitig mit dem PAPER-Befehl verändert und besitzt den eigenen Befehl

BORDER farbe

„farbe" bezeichnet den tatsächlichen Farbwert (0-26) und nicht die Faßnummer! Wie bei INK ist es ebenfalls möglich, den Border blinken zu lassen.

BORDER farbe1 , farbe2

2.5 Blinkgeschwindigkeit - SPEED INK

Durch SPEED INK ist die Einschaltdauer der alternativen Farben (INK oder BORDER) regelbar.

SPEED INK zeit1 , zeit2

Die Parameter „zeit1" und „zeit2" bestimmen die Einschaltzeit der alternativen Farben („farbe1", „farbe2").

SPEED INK 20, 10

Die „farbe1" erscheint 20 Einheiten zu je 0.02 Sekunden und die „farbe2" 10 Einheiten zu je 0.02 Sekunden (20*0.02 = 400 mSec, 10*0.02 = 200 mSec). Im Einschaltzustand (Reset) wechseln die Farben alle 200 mSec (entspricht SPEED INK 10,10).

```
1000 REM ************************************ M.W.Thoma **
1010 REM *                                               *
1020 REM *                    SPEED INK                  *
1030 REM *                                               *
1040 REM *************************************************
1050 MODE 1
1060 INK 1,6,24
1070 LOCATE 10,10
1080 PEN 1
1090 PRINT "Blinkzeit in Sekunden!
1100 FOR i=1 TO 50
1110 SPEED INK i,i
1120 PEN 2
1130 LOCATE 15,12
1140 PRINT "Zeit :";USING "##.###";i*0.02
1150 LOCATE 12,15
1160 PRINT "SPEED INK ";i;",";i
1170 FOR t=1 TO 2500:NEXT t
1180 PRINT CHR$(7)
1190 NEXT i
1200 REM *************************************************
```

Das Blinken der Zeichen wird durch einen Timer (Zeitgeber) durchgeführt, so daß alle Farben im selben Rhythmus blinken.

2.6 Transparent-Modus

Normal wird ein Zeichen auf den Bildschirm gePRINTet, indem ein Feld von 64 Punkten (siehe 4.1) auf den Bildschirm gelegt und jeweils als gesetzter oder gelöschter Punkt erscheint. Ein gesetzter Punkt wird in der Zeichenfarbe und ein gelöschter Punkt in der PAPER-Farbe interpretiert, was zur Folge hat, daß ein neues Zeichen ein anderes komplett überschreibt. Besonders störend ist dieses, wenn Grafiken beschriftet werden sollen, da ein Teil der Grafik (64 Punkte) einfach überschrieben wird. Mit der Möglichkeit, Zeichen transparent (durchscheinend) darzustellen, bleibt der Hintergrund weitestgehend erhalten, da der Hintergrund (z.B. eine Linie) durch das neue Zeichen scheint. Der Transparent-Modus wird mit

PRINT CHR$(22)+CHR$(1)

eingeschaltet und bleibt solange aktiv, bis ein

PRINT CHR$(22)+CHR$(0)

folgt.

Das nachfolgende Demonstrationsprogramm zeigt deutlich die Unterschiede. In einer Grafik (Linien) sind zwei Sätze geschrieben worden, wobei beim ersten Satz der Transparent-Modus eingeschaltet ist.

```
1000 REM ************************************ M.W.Thoma **
1010 REM *                                               *
1020 REM *              Transparents-Demo                *
1030 REM *                                               *
1040 REM *************************************************
1050 MODE 1
1060 FOR x=0 TO 639 STEP 18
1070 MOVE 0,0
1080 DRAW x,399,1
1090 NEXT
1100 PRINT CHR$(22)+CHR$(1)
1110 LOCATE 5,5
1120 PEN 3
1130 PRINT"Dieser Text ist transparent"
1140 PRINT CHR$(22)+CHR$(0)
```

```
1150 LOCATE 5,7
1160 PRINT"Und dieser nicht mehr"
1170 FOR t=1 TO 3000:NEXT t
1180 FOR f=2 TO 26: INK 3,f
1190 FOR t=1 TO 1000:NEXT t
1200 NEXT f
1210 END
1220 REM ***************************************************
```

2.7 Inverse-Modus

Um bestimmte Bildschirmausgaben besonders hervorzuheben (z.B. Bedienungsfehler, Zahlen), bietet sich die „Inverse-Darstellung" an. Das Invertieren eines Zeichens bewirkt, daß ein gesetzter Zeichenpunkt nicht mehr in der Zeichenfarbe, sondern in der Hintergrundfarbe erscheint. Ein gelöschter Zeichenpunkt erscheint dafür in der Zeichenfarbe. Wie beim Transparent-Modus wird der Inverse-Modus mit einem Steuercode (Steuerzeichen) aktiviert:

PRINT CHR$(24)

und bleibt solange aktiv, bis wiederum ein „PRINT CHR$(24)" folgt. Das bedeutet, das ein „PRINT CHR$(24)" die inverse Darstellung entweder ein- oder aber ausschaltet, abhängig vom momentan gewählten Modus. In der Praxis könnte dieses folgendermaßen aussehen:

```
1000 REM ************************************ M.W.Thoma **
1010 REM *                                               *
1020 REM *                Inverse - Modus                *
1030 REM *                                               *
1040 REM ***************************************************
1050 MODE 2
1060 PRINT CHR$(24)
1070 PRINT "  Inverse-Modus ist und bleibt aktiviert  "
1080 PRINT:PRINT
1090 FOR i= 1 TO 140
1100 PRINT USING "####";i,
1110 NEXT i
1120 PRINT:PRINT
1130 PRINT"  bis ein weiteres 'CHR$(24)' ausgegeben wird."
1140 PRINT CHR$(24)
1150 PRINT
1160 FOR i= 1 TO 140
1170 PRINT USING "####";i,
1180 NEXT i
1190 END
1200 REM ***************************************************
```

3.0 Bildschirmspeicher

Der Bildschirmspeicher ist der Bereich innerhalb des 64 kByte großen RAM-Speicherbereichs, in dem der Inhalt des Bildschirms abgelegt ist. Alle Zeichen, Symbole oder Linien, die auf dem Bildschirm sichtbar sind, werden hier als Bitmuster abgelegt. Jeder der 128000 Bildschirmpunkte (640*200) ist einem Bit zugeordnet, welches gesetzt ('1') oder gelöscht ('0') sein kann. Je 8 Bits (Bildpunkte) sind als Byte zusammengefaßt und bilden einen Bildschirmspeicherplatz:

128000 / 8 = 16000 Bytes

Der gesamte Bildschirmspeicher benötigt also einen Speicherplatz von exakt 16000 Bytes. Innerhalb des vorhandenen 64 kByte großen RAM-Speicherbereichs müssen mindestens 16000 Bytes für den Bildschirmspeicher reserviert und vor dem Überschreiben durch Programme und Daten geschützt werden. Einfachheitshalber ist der Bildschirmspeicher mit 16 kByte (16384 Bytes) ausgelegt worden und befindet sich (normalerweise) am Ende des 64 kByte RAM-Speichers in den Adressen 49152 bis 65535 ($C000 - $FFFF).

Somit steht nur der Bereich bis Adresse 49151 ($BFFF) für Programme und Daten frei zur Verfügung (auch von diesem Bereich geht noch einiger Speicherplatz für Betriebsroutinen verloren).

Innerhalb des Bildschirmspeichers stehen die Informationen über

a) die Form der darzustellenden Zeichen und
b) die Farbe, in der ein Punkt erscheinen soll.

3.1 Organisation des Bildschirmspeichers

Die erste Speicherstelle des Bildschirmspeichers (49152) repräsentiert nach einem CLS- oder MODE-Befehl 8 Bildschirmpunkte in der oberen, linken Ecke (die folgende Betrachtung bezieht sich immer auf MODE 2 !). Mit

```
MODE 2: POKE 49152, 255
```

wird diese Speicherstelle mit dem dezimalen Wert 255 geladen und es erscheint ein Strich in einer Länge von 8 Punkten.

11111111 = 255

Die nachfolgende Speicherstelle (49153) liegt nun direkt daneben:

```
POKE 49153, 255
```

Alle anderen nachfolgenden Bytes liegen wiederum rechts daneben, bis das Ende der Zeile erreicht wird (80 Byte).

```
10 MODE 2
20 FOR sp=49152 TO 49152+79
30 POKE sp, 255
40 NEXT sp
50 END
```

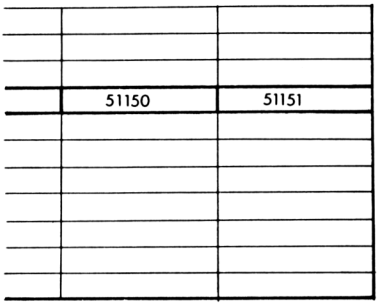
Das kleine Programm zeichnet eine Linie in der obersten Zeile von links nach rechts. Die nächsten 80 Bytes des Bildschirmspeichers (49232 - 49311) bestimmen nun die Bildschirmpunkte, die genau 8 Punkte unterhalb der ersten Linie liegen. Sehen Sie sich das einmal an, indem Sie die Programmzeile 20 ändern:

```
20 FOR sp =49152 TO 49152+159
```

Es ist deutlich zu erkennen, daß der zweite Teil dieser Linie 8 Punkte tiefer liegt und der Bildschirmspeicher wie folgt organisiert ist:

49152	49153	49154	
49232	49233	49234	
49312	49313	49314	

Dieses Schema setzt sich so bis in die untere, rechte Ecke fort.



Ändern Sie noch einmal die Programmzeile 20 und sehen Sie sich das Ergebnis an:

```
20 FOR sp=49152 TO 49152+1999
```

Das Programm zeichnet jetzt die „1. Reihe" von der oberen, linken in die untere, rechte Ecke.
Die „2. Reihe" beginnt nun wieder in der oberen, linken Ecke, allerdings einen Punkt tiefer als die „1. Reihe" und entspricht der Speicherstelle 49152+2048 (!!!!). Sie werden sich sicherlich fragen, was mit den Bytes 2000-2047 des Bildschirmspeichers ist? Diese 48 Bytes am Ende einer „Reihe" werden normalerweise nicht benutzt! Die „2. Reihe" beginnt also mit Byte 2048 und endet im Byte 4047 (2048+1999). Nun folgen wieder 48 unbenutzte Bytes und die „3. Reihe" beginnt (oben links) mit Byte 4046. Dieses Schema setzt sich nun bis zum Ende der „8. Reihe" fort.

```
20 FOR sp=49152 TO 65535-48
```

Ist die Programmzeile 20 wie oben geändert, wird der gesamte Bildschirm mit Punkten gefüllt (die letzten 48 Bytes der „8. Reihe" brauchen nicht gesetzt zu werden, da sie nicht benutzt sind).

49152	49153	
51200	51201	
53248	53249	
55296	55297	
57344	57345	
59392	59393	
61440	61441	
63488	63489	

Bitte beachten Sie eine kleine Besonderheit des Bildschirmaufbaus: sowie der Bildschirm nach oben oder unter scrollt (rollt), verschiebt sich der Bildschirmspeicher entsprechend!! Führen Sie bitte einmal folgenden Versuch durch:

MODE 2
(Den Cursor nach oben bewegen, bis der Bildschirm scrollt.)
POKE 49152, 255

Die Speicherstelle 49152 repräsentiert jetzt nicht mehr die obere, linke Ecke (!!!), sondern den Ort, bevor das Scrolling stattgefunden hat! Beachten Sie diese Besonderheit, falls Sie direkt auf den Bildschirmspeicher Einfluß nehmen wollen und verhindern Sie in diesem Fall das Scrollen.

3.2 Farbe und Form

Die bisherige Betrachtung des Bildschirmspeichers bezog sich ausschließlich auf den 80-Zeichen Modus (MODE 2). Der Modus 2 bietet die größtmögliche Auflösung von 640*200 direkt ansprechbaren Bildpunkten, wobei allerdings nur 2 verschiedene Farben gleichzeitig darstellbar sind.
Ein gesetztes Bit des Bildschirmspeichers wird in der Farbe aus dem Farbfaß 1 (Zeichenfarbe) und ein gelöschtes Bit in der Farbe aus Faß 0 (Hintergrundfarbe) interpretiert.

Mit „MODE 1" wird auf den 40 Zeichen-Modus umgeschaltet. Ein Mode-Befehl bewirkt immer, daß gleichzeitig der Bildschirm gelöscht (CLS) wird und alle WINDOWs aufgehoben werden (siehe 5.0).
In diesem Modus beträgt die Bildschirmauflösung „nur" noch 320*200 einzeln ansprechbarer Bildpunkte. Die Halbierung der X-Achsenauflösung resul-

tiert aus der Farbinformation, die zusätzlich im Bildschirmspeicher untergebracht werden muß. Im Gegensatz zum 80 Zeichenmodus (MODE 2) repräsentieren 2 Bits jeweils einen Punkt, der nun allerdings in 4 verschiedenen Farben erscheinen kann. Je nach Kombination der Bitpaare wird die Punktfarbe aus einem der 4 möglichen Farbfässer geholt:

00 = Farbe aus Faß 0
01 = Farbe aus Faß 1
10 = Farbe aus Faß 2
11 = Farbe aus Faß 3

Der Bildschirmspeicher wird nicht mehr bitweise sondern paarweise interpretiert, so daß immer 2 (echte) Bildpunkte in der gleichen Farbe gesetzt werden. Somit ist ein Bildschirmbyte nur noch für 4 (doppelte) Punkte verantwortlich. Leider liegen die zusammengehörigen Bits innerhalb des Bytes nicht direkt nebeneinander. Jedes Byte des Bildschirmspeichers hat im Modus 1 nachfolgenden Aufbau:

Bits	76543210
1.Bitpaar	X...X...
2.Bitpaar	.X...X..
3.Bitpaar	..X...X.
4.Bitpaar	...X...X

Die zusammengehörigen Bits liegen innerhalb des Bytes also an den Stellen:

3/7 2/6 1/5 0/4

Ein Beispiel soll noch einmal den Zusammenhang zwischen Bits und Farbe verdeutlichen: 4 Bildschirmpunkte sollen in den 4 möglichen Farben gesetzt werden.

Faß 0 = 00	Bits 3/7	0...0...
Faß 1 = 01	Bits 2/6	.1...0..
Faß 2 = 10	Bits 1/5	..0...1.
Faß 3 = 11	Bits 0/4	...1...1
		01010011

01010011 = 83

MODE 1: POKE 49152, 83

In der oberen, linken Ecke sind 4 farbige Punkte zu erkennen. Der 1. Punkt ist nicht sichtbar, da er in der Hintergrundfarbe gesetzt ist.

Der Modus 0 (MODE 0) ist der sogenannte Vielfarb- oder Multicolor-Modus. Hier sind immerhin 16 verschiedene Farben gleichzeitig darstellbar. Natürlich hat das Auswirkungen auf die Auflösung: Sie beträgt nur noch 160*200 einzeln ansprechbarer Bildpunkte.
Um 16 Farben unterscheiden zu können, benötigt der Rechner 4 Bits (auch „Nibble" genannt). Je nach Kombination der zusammengehörigen 4 Bits erscheint der Punkt in der entsprechenden Farbe.

0000	:	Farbe aus Faß 0
0001	:	Farbe aus Faß 1
0010	:	Farbe aus Faß 2
0011	:	Farbe aus Faß 3
0100	:	Farbe aus Faß 4
0101	:	Farbe aus Faß 5
0110	:	Farbe aus Faß 6
0111	:	Farbe aus Faß 7
1000	:	Farbe aus Faß 8
1001	:	Farbe aus Faß 9
1010	:	Farbe aus Faß 10
1011	:	Farbe aus Faß 11
1100	:	Farbe aus Faß 12
1101	:	Farbe aus Faß 13
1110	:	Farbe aus Faß 14
1111	:	Farbe aus Faß 15

Auch in diesem Modus liegen die 4 zusammengehörigen Bits leider nicht direkt nebeneinander, sondern wieder verteilt:

linker Punkt	:	Bits 1/5/3/7
rechter Punkt	:	Bits 0/4/2/6

Es werden jeweils 4 Bildpunkte in der gleichen Farbe gesetzt, so daß nur noch 2 „echte" Punkte vorhanden sind.
Je nach gewähltem Bildschirm-Modus repräsentiert ein Byte im

MODE 0	2 echte Punkte
MODE 1	4 echte Punkte
MODE 2	8 echte Punkte

Abschließend noch ein Beispiel zur Farbgebung im Modus 0:

Faß 1 : 0001	Bits 1/5/3/7 :	1.0.0.0.
Faß 2 : 0010	Bits 0/4/2/6 :	.0.0.1.0
		10000100

10000100 = 132

MODE 0: POKE 49152, 132

3.3 „Garfield" in Multicolour

Sicherlich kennen Sie „Garfield" – den faulsten Kater der Welt. Die Katze ist in einzelne, farbige Punkte zerlegt und entsprechend der vorhergehenden Beschreibung in Bytes umgerechnet. Es ist schon eine recht mühselige Arbeit, aber das Ergebnis läßt sich sehen: Schauen Sie selber.

```
1000 REM ********************************** M.W.Thoma **
1010 REM *                                             *
1020 REM *        Multicolor-Grafik im MODE 0          *
1030 REM *                                             *
1040 REM *             G A R F I E L D                 *
1050 REM *                                             *
1060 REM ***********************************************
1070 MODE 0
1080 INK 0,24:INK 1,0:INK 2,15:INK 3,11
1090 INK 4,26:INK 5,24:INK 6,6,13
1100 PAPER 1:BORDER 0
1110 CLS
1120 FOR i= 49900 TO 50220 STEP 80
1130 FOR Y=0 TO 16383 STEP 2048
1140 FOR j=0 TO 23
1150 READ x
1160 IF x=-1 THEN 1210
1170 POKE i+j+Y,x
1180 NEXT J
1190 NEXT Y
1200 NEXT I
1210 PEN 7
1220 LOCATE 7,6
1230 PRINT"Garfield"
```

```
1240 LOCATE 7,17
1250 PRINT"the king"
1260 GOTO 1260
1270 '
1280 REM ************************************************
1290 REM *                                              *
1300 REM *          Daten fuer die Grafik               *
1310 REM *                                              *
1320 REM ************************************************
1330 DATA 000,000,000,000,000,000,000,000,192,192,000,064
1340 DATA 192,192,000,000,000,000,000,000,000,000,000,000
1350 DATA 000,000,000,000,000,000,000,064,192,192,192,132
1360 DATA 012,132,192,000,000,000,000,000,000,000,000,000
1370 DATA 000,000,000,000,000,000,000,192,192,192,072,132
1380 DATA 132,072,012,128,000,000,000,000,000,000,000,000
1390 DATA 000,000,000,000,000,000,000,192,192,192,012,192
1400 DATA 072,012,132,072,128,000,000,000,000,000,000,000
1410 DATA 000,000,000,000,000,000,000,192,012,192,132,072
1420 DATA 012,192,192,012,128,000,000,000,000,000,000,000
1430 DATA 000,000,000,000,000,000,000,132,132,132,072,012
1440 DATA 012,012,132,072,072,000,000,000,000,000,000,000
1450 DATA 000,000,000,000,000,000,000,064,012,072,132,132
1460 DATA 072,192,192,072,132,128,000,000,000,000,000,000
1470 DATA 000,000,000,000,000,000,000,000,192,196,200,192
1480 DATA 196,204,204,132,132,072,128,000,000,000,000,000
1490 DATA 000,000,000,000,000,000,000,132,204,204,204,196
1500 DATA 204,204,204,200,012,012,072,128,000,000,000,000
1510 DATA 000,000,000,000,000,000,064,072,204,204,204,196
1520 DATA 204,204,204,204,132,012,012,072,000,000,000,000
1530 DATA 000,000,000,000,000,000,132,196,204,204,200,204
1540 DATA 204,204,204,204,132,012,012,132,128,000,000,000
1550 DATA 000,000,000,000,000,064,012,196,204,204,200,204
1560 DATA 204,204,204,200,132,012,012,072,192,000,000,000
1570 DATA 000,000,000,000,000,064,012,072,192,192,192,192
1580 DATA 192,192,192,192,192,012,012,012,072,128,000,000
1590 DATA 000,000,000,000,000,132,012,144,096,192,096,048
1600 DATA 192,144,048,048,096,208,132,012,132,072,000,000
1610 DATA 000,000,000,000,000,132,072,208,048,048,096,144
1620 DATA 048,048,048,048,208,240,224,012,012,072,000,000
1630 DATA 000,000,000,000,064,012,208,240,192,192,148,104
1640 DATA 048,048,048,096,208,240,240,132,072,072,128,000
1650 DATA 000,000,000,000,064,012,208,240,208,240,148,104
1660 DATA 192,192,192,208,224,240,240,132,012,012,128,000
1670 DATA 000,000,000,000,064,072,240,240,208,224,072,208
1680 DATA 240,240,240,224,208,240,240,224,012,132,128,000
1690 DATA 000,000,000,000,064,012,028,240,224,132,012,072
1700 DATA 208,240,224,132,072,240,240,132,012,012,128,000
1710 DATA 000,000,000,000,000,132,208,240,224,012,012,012
1720 DATA 072,192,132,012,012,208,240,132,012,012,128,000
1730 DATA 000,192,000,000,000,132,072,240,192,012,012,012
1740 DATA 012,012,012,012,012,072,192,012,012,012,128,000
1750 DATA 064,012,192,000,000,132,012,192,012,012,012,012
1760 DATA 012,012,012,012,012,132,012,012,012,072,000,000
1770 DATA 064,012,012,128,000,064,012,012,072,012,012,012
1780 DATA 012,012,012,012,012,012,012,012,012,072,000,000
```

```
1790 DATA 000,132,012,072,000,000,132,012,072,012,012,012
1800 DATA 012,012,012,012,132,012,012,012,012,128,000,000
1810 DATA 000,064,012,072,192,000,064,012,132,012,012,012
1820 DATA 012,012,012,012,072,012,012,012,012,128,000,000
1830 DATA 000,192,192,132,012,128,000,192,192,012,012,012
1840 DATA 012,012,012,012,012,192,132,012,072,000,000,000
1850 DATA 064,012,072,012,012,072,000,000,000,132,012,012
1860 DATA 192,192,192,192,072,012,192,192,072,000,000,000
1870 DATA 064,192,192,132,072,012,192,192,192,132,012,012
1880 DATA 012,012,012,012,012,012,012,012,012,128,000,000
1890 DATA 132,012,132,012,072,132,012,012,072,012,012,012
1900 DATA 012,012,012,012,012,012,012,012,072,000,000,000
1910 DATA 064,012,132,012,012,012,012,012,132,012,012,012
1920 DATA 012,012,012,012,012,012,012,012,012,128,000,000
1930 DATA 000,192,192,012,012,132,012,072,012,012,012,012
1940 DATA 012,012,012,012,012,012,132,012,012,072,000,000
1950 DATA 000,000,000,132,072,192,012,072,012,012,012,012
1960 DATA 012,012,012,012,012,012,132,012,012,072,000,000
1970 DATA 000,000,000,064,128,000,192,132,012,012,012,012
1980 DATA 012,012,072,192,012,072,012,012,012,012,128,000
1990 DATA 000,000,000,000,000,000,064,132,012,012,012,012
2000 DATA 012,012,132,012,132,132,012,012,012,012,072,000
2010 DATA 000,000,000,000,000,000,064,012,012,012,012,012
2020 DATA 012,072,012,012,072,012,012,012,132,012,012,128
2030 DATA 000,000,000,000,000,000,064,012,012,012,012,012
2040 DATA 012,132,012,012,012,012,012,072,012,012,012,128
2050 DATA -1
2060 REM ************************************************
```

4.0 Der Zeichensatz

Insgesamt stehen im CPC-464 256 verschiedene Zeichen (Buchstaben, Zahlen und Symbole) zur Verfügung. Über die Tastatur sind allerdings nur 128 dieser Zeichen abrufbar. 32 „Zeichen" (Code 0-32) sind sogenannte Steuerzeichen (siehe 16.0), die eine bestimmte Funktion ausführen (z.B. Transparent-Modus einschalten) und normalerweise kein Symbol auf dem Bildschirm ausgeben. Der Standard-Zeichensatz ist sehr gut ausgewählt worden. So sind einige griechische Buchstaben, das 1/4-, 1/2- und 3/4-Zeichen sowie andere nützliche Symbole enthalten. Leider hat es auch die Firma Schneider versäumt, eine deutsche DIN-Tastatur mit deutschem Zeichensatz auf dem Markt anzubieten.

Alle Zeichen müssen einem dem Computer verständlichen binären Code zugeordnet sein. In Amerika wurde der ASCII-Code (sprich Aski) entwickelt. ASCII steht für „American Standart Code for Information Interschange". Der ASCII-Code arbeitet standardmäßig mit einer Wortbreite von 7 Bits, so daß insgesamt 128 Zeichen ansprechbar sind. Beim CPC-464 stehen neben den 128 Standard-ASCII-Zeichen (0-127) noch 128 weitere Zeichen zur Verfügung (128-255).
Sehen Sie sich einmal alle „Schneider-Zeichen" an:

```
10 FOR i= 0 to 255
20 PRINT CHR$(1);CHR$(i);
30 NEXT i
40 END
```

Das vorangestellte CHR$(1) bewirkt, daß die Steuerzeichen (Code 0-31) als Symbole ausgegeben werden und nicht ihre Funktion ausgeführt wird.
Mit den vielen Grafik-Zeichen und Symbolen, die fest im Zeichensatz enthalten sind, lassen sich kleine Grafiken erstellen. Gerade in der Animation (Einführung in Spielprogramme etc.) bietet sich diese einfache Form der Grafikgestaltung an. Hierzu zwei kleine Beispiele:

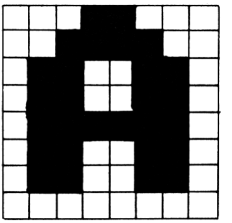
```
1000 REM ********************************** M.W.Thoma **
1010 REM *                                             *
1020 REM *                Beispiel Nr. 1               *
1030 REM *                                             *
1040 REM ***************************************************
1050 MODE 0
1060 INK 1,20:INK 2,15
1070 t$="Manfred Walter Thoma"
1080 FOR i= 1 TO 20
```

```
1090 PEN 2
1100 FOR j= 248 TO 251
1110 LOCATE i,12
1120 PRINT CHR$(j)
1130 FOR t = 1 TO 50:NEXT t
1140 NEXT j
1150 SOUND 1,1200,10,7
1160 PEN 1
1170 LOCATE i,12
1180 PRINT MID$(t$,i,1)
1190 NEXT i
1200 FOR t=1 TO 2000:NEXT t
1210 REM ************************************************
1220 '
1230 REM ************************************************
1240 REM *                                              *
1250 REM *                Beispiel Nr. 2                *
1260 REM *                                              *
1270 REM ************************************************
1280 SPEED INK 5,5
1290 PEN 3
1300 t$="    praesentiert    "
1310 FOR i= 1 TO 20
1320 FOR j= 1 TO 12
1330 LOCATE i,j
1340 PRINT CHR$(252)
1350 FOR t= 1 TO 50:NEXT t
1360 SOUND 1,200-j*12,4,5
1370 LOCATE i,j
1380 PRINT CHR$(32)
1390 NEXT j
1400 LOCATE i,12
1410 PEN 15
1420 PRINT CHR$(238)
1430 SOUND 1,1000,50,7,1,1,1
1440 FOR t= 1 TO 100:NEXT t
1450 LOCATE i,12
1460 PEN 3
1470 PRINT MID$(t$,i,1)
1480 NEXT i
1490 LOCATE 1,25
1500 FOR i=1 TO 12
1510 PRINT
1520 FOR t=1 TO 100:NEXT t
1530 NEXT i
1540 END
1550 REM ************************************************
```

4.1 Das Zeichen-ROM

Sieht man sich ein Zeichen (z.B. „A“) genauer an, so kann man folgendes Muster erkennen:

```
Byte 0 : 00011000  24
Byte 1 : 00111100  60
Byte 2 : 01100110 102
Byte 3 : 01100110 102
Byte 4 : 01111110 126
Byte 5 : 01100110 102
Byte 6 : 01100110 102
Byte 7 : 00000000   0
```



Ein Zeichen besteht aus 64 Punkten (8*8). Jede Zeile entspricht einem Byte. Betrachtet man die gesetzten Punkte als binäre '1', so erhält man einen entsprechenden Wert.
Innerhalb des Betriebssystems ist der gesamte Zeichensatz Byte für Byte abgelegt. Für seine 256 Zeichen benötigt er einen Speicherraum von

Speicherraum = 256 Zeichen * 8 Byte
Speicherraum = 2048 Byte
Speicherraum = 2 kByte

Der Zeichensatz liegt am Ende des Betriebssystems in den Adressen 14336 bis 16834 ($3800 - $3FFF).

Das Betriebssystem befindet sich genau wie der Zeichensatz in einem ROM, welches dem RAM-Bereich 0 - 16834 überlagert ist. Im Zeichen-ROM selber liegen die Werte für alle Zeichen hintereinander: Als erstes Zeichen ASCII 0, dann ASCII 1 bis ASCII 255.

Das Arbeiten mit dem vorhandenen Zeichensatz hat dort seine Grenze, wo Zeichen benötigt werden, die nicht im Zeichensatz vorhanden sind. So z.B. die deutschen Umlaute und Sonderzeichen (ä,ü,ö,ß). Damit aber Einfluß auf die Form der Zeichen genommen werden kann, ist es notwendig, das Zeichen-ROM in einen RAM-Bereich zu kopieren.

4.2 SYMBOL AFTER - Zeichensatz kopieren

Das Kopieren des Zeichensatzes aus dem ROM in einen RAM-Bereich ist fest im Schneider-BASIC integriert. Mit

SYMBOL AFTER x

wird ein bestimmter Teil des Zeichensatzes kopiert. 'x' ist ein ganzzahliger Ausdruck zwischen 0 und 256 (!).
Um die ASCII-Zeichen 60 - 255 ins RAM zu kopieren, muß der Befehl

```
SYMBOL AFTER 60
```

eingegeben werden. Alle Zeichen werden mit

```
SYMBOL AFTER 0
```

kopiert.

Es taucht sofort die Frage auf, wohin der Zeichensatz genau kopiert wird? Die Antwort ist einleuchtend: an das Ende des BASIC-RAM-Bereichs. Aber prüfen Sie doch einmal selber nach. Führen Sie einen Reset (CTRL/SHIFT/ESC) durch und fragen mit HIMEM nach dem BASIC-RAM-Ende (Da bei Benutzung der Diskettenstation DDI-1 oder des CPC-664 1284 Bytes zusätzlich benötigt werden, ist das HIMEM entsprechend nach unten verschoben (siehe 1.0). Alle in Klammern stehenden Werte beziehen sich auf den Speicheraufbau bei angeschlossener Diskettenstation bzw. des CPC-664.).

```
PRINT HIMEM          PRINT HIMEM
 43903                42619
```

Das Ergebnis 43903 (42619) zeigt die letzte für Programme und Daten verfügbare Speicherstelle an. Kopieren Sie nun alle Zeichen ins RAM

```
SYMBOL AFTER 0
```

und lassen sich noch einmal das BASIC-RAM-Ende anzeigen:

```
PRINT HIMEM          PRINT HIMEM
 41983                40699
```

Das BASIC-RAM-Ende hat sich nach unten verschoben und oberhalb der Adresse 41983 (40699) liegt der kopierte Zeichensatz in einem geschützten RAM-Bereich. Gleichzeitig hat sich der verfügbare Speicherraum von 43533 (42249) freien Bytes auf

```
PRINT FRE (0)        PRINT FRE (0)
 41613                40329
```

reduziert.
Doch beachten Sie bitte folgendes: nach der Eingabe von

SYMBOL AFTER 256 (!!!)

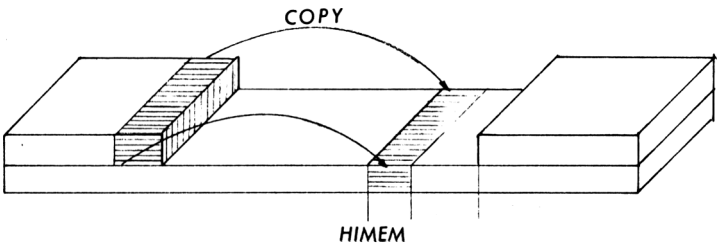
(was bedeutet, daß keine Zeichen kopiert werden sollen) steht plötzlich mehr freier Speicherplatz zur Verfügung!!

PRINT HIMEM	PRINT HIMEM
44031	42747

Die Adresse 44031 (42747) ist nun tatsächlich das BASIC-RAM-Ende. Das nach einem „SYMBOL AFTER 256" mehr Speicherplätze als nach einem Reset (Einschaltzustand) zur Verfügung stehen, liegt an der Tatsache, daß das Betriebssystem nach einem Reset die letzten 16 Zeichen (ASCII 240-255) automatisch ins RAM kopiert. Rechnen Sie es einmal nach:

BASIC-RAM-Ende	:	44031	42747
16 Zeichen a 8 Byte	:	– 128	– 128
neues BASIC-RAM-Ende	:	43903	42619

Das Ergebnis 43903 (42619) ist genau die Speicherstelle, die auch nach einem Reset als letzte BASIC-RAM-Speicherstelle ausgegeben wird.



Nachdem die Zeichen aus dem ROM- in den RAM-Bereich kopiert worden sind, lassen sie sich ohne weiteres auslesen und gegebenenfalls verändern.

4.3 Zeichen auslesen

Nach dem Kopieren aller Zeichen (SYMBOL AFTER 0) befindet sich das 1. Byte des Zeichensatzes in der Adresse 41984 (40700). Dieses 1. Byte entspricht dem 1. Byte des ASCII-Zeichens 0. Die darauffolgenden 7 Bytes sind die Bytes 1 - 7 für das ASCII-Zeichen 0. Dann folgen die 8 Bytes für das Zeichen ASCII 0 usw. bis zum letzten Byte von ASCII 255. Das erste Byte eines Zeichens läßt sich daraufhin allgemein mit der Gleichung

1.Byte des Zeichens = 41984+8*ASCII (=40700+8*ASCII)

bestimmen. Hierzu ein Beispiel. Sie suchen das 1. Byte des ASCII-Zeichens 65 (='A'):

1.Byte = 41984+8*65 =40700+8*65
1.Byte = 42504 1220

Die auf Adresse 42504 (41220) folgenden 7 Bytes sind die restlichen Bytes des Zeichens 'A'. Mit einer kleinen Schleife lassen sich dann alle Bytes auslesen:

```
10 MODE 1
20 SYMBOL AFTER 0
30 INPUT "ASCII-Code (0-255)";ascii
40 FOR i=0 TO 7
50 PRINT PEEK (41984+8*ascii+i
(  50 PRINT PEEK (40700+8*ascii+i
60 NEXT i
70 END
```

Mit dem nachfolgenden Programm zum Auslesen der Zeichen können Sie sich den Aufbau eines Zeichens wesentlich genauer ansehen.
Nach dem Bildschirmaufbau (Zeilen 1000-1160), folgt die Eingabe des gewünschten Zeichens (1230). Bei einer Eingabe kleiner als 0 oder größer als 255 wird das Programm beendet (1240). In den Zeilen 1250-1270 finden Sie das Einlesen der Zeichenwerte wieder. In der Schleife 1320-1350 wird für jedes gesetzte Bit ein inverses SPACE oder der Dezimalpunkt für ein gelöschtes Bit ausgegeben. Anschließend wird noch der binäre und dezimale Wert des Bytes ausgegeben.

```
1000 REM ************************************ M.W.Thoma **
1010 REM *                                               *
1020 REM *                Zeichen auslesen               *
1030 REM *                                               *
1040 REM *************************************************
1050 DIM adr(7)
1060 SYMBOL AFTER 0
1070 start=HIMEM+1
1080 MODE 1
1090 INK 0,0:INK 1,15:INK 2,20:BORDER 0
1100 PRINT "        Auslesen des Zeichensatzes"
1110 LOCATE 5,5
1120 PRINT"Byte   ";
1130 FOR i= 7 TO 0 STEP-1
1140 PRINT USING "#";i;
1150 NEXT i
1160 PRINT "    BINAER   DEZ"
1170 '
1180 REM *************************************************
1190 REM *                                               *
1200 REM *              Eingabe und Anzeige              *
1210 REM *                                               *
1220 REM *************************************************
1230 LOCATE 5,21
1240 INPUT "Welches Zeichen (0-255) :";z
1250 IF z<0 OR z>255 THEN MODE 1:END
1260 FOR s=0 TO 7
1270 adr(s)=PEEK(start+z*8+s)
1280 NEXT s
1290 FOR s=0 TO 7
1300 LOCATE 4,7+s
1310 PRINT USING "#####";start+z*8+s;:PRINT "   ";
1320 PEN 2
1330 FOR bi=7 TO 0 STEP-1
1340 IF (adr(s) AND 2^bi) <> 2^bi THEN PRINT ".";:GOTO 1360
1350 PRINT CHR$(143);
1360 NEXT bi
1370 PEN 1
1380 LOCATE 23,7+s
1390 PRINT BIN$(adr(s),8)
1400 LOCATE 33,7+s
1410 PRINT USING "###";adr(s)
1420 NEXT s
1430 GOTO 1230
1440 REM *************************************************
```

4.4 SYMBOL - neue Zeichen erstellen

Nach dem Kopieren des Zeichensatzes bzw. eines Teils des Zeichensatzes (SYMBOL AFTER) gibt es zwei Möglichkeiten, Zeichen neu zu definieren:

a) direkt Einfluß auf den Zeichensatz nehmen (POKE) oder
b) den BASIC-Befehl „SYMBOL".

Mit dem Befehl „SYMBOL" ist es möglich, jedes der 256 Zeichen neu zu bestimmen:

SYMBOL ascii, byte0, byte1, byte2, byte3, ... byte7

Nach SYMBOL folgt der ASCII-Code des zu ändernden Zeichens, dann 8 Bytes, die die neue Form des Zeichens bestimmen. Sind weniger als 8 Bytes angegeben, werden die fehlenden Werte als '0' angenommen und dem Zeichen als '0' übergeben. Mindestens muß ein Byte angegeben werden.

Jedes gesetzte Bit ('1') eines Bytes wird als gesetzter und jedes gelöschte Bit ('0') als nicht gesetzter Punkt interpretiert.

Byte 0		00011000	24
Byte 1		00111100	60
Byte 2		01100110	102
Byte 3		11000011	195
Byte 4		00000000	0
Byte 5		11111111	255
Byte 6		00000000	0
Byte 7		00000000	0

Das Entspricht-Zeichen ist hier in seine einzelnen Punkte (Bits) zerlegt und als dezimale Zahl zusammengefaßt worden. Nun ist noch zu entscheiden, auf welches ASCII-Zeichen das neue Zeichen zu legen ist, z.B. auf den ASCII-Code 240:

SYMBOL 240, 24, 60, 102, 195, 0, 255, 0, 0

Mit diesem Verfahren lassen sich alle Zeichen neu definieren. Interessant ist auch die Möglichkeit, mehrere Zeichen zu einer „Kleingrafik" zusammenzufassen. Ein Zeichen stellt dann nur einen Teil des gesamten Zeichens dar. Als Beispiel ist das „BTX-Zeichen" aus 6 einzelnen Zeichen zusammengesetzt worden (250-255).

```
1000 REM ************************************ M.W.Thoma **
1010 REM *                                               *
1020 REM *                  BTX - Zeichen                *
1030 REM *                                               *
1040 REM ****************************************************
1050 SYMBOL 250,63,64,159,191,188,152,128,140
1060 SYMBOL 251,255,0,255,255,0,60,66,189
```

```
1070 SYMBOL 252,252,2,241,249,121,49,1,49
1080 SYMBOL 253,156,188,190,191,191,159,64,63
1090 SYMBOL 254,189,66,60,0,255,255,0,255
1100 SYMBOL 255,57,61,125,253,253,249,2,252
1110 MODE 1
1120 LOCATE 19,10
1130 PRINT CHR$(250);CHR$(251);CHR$(252)
1140 LOCATE 19,11
1150 PRINT CHR$(253);CHR$(254);CHR$(255)
1160 LOCATE 14,13
1170 PRINT"BTX - Zeichen"
1180 GOTO 1180
1190 REM ***************************************************
```

Die zweite Variante, direkt auf den Zeichensatz Einfluß zu nehmen, bietet ebenfalls interessante Möglichkeiten. Nachdem der gesamte Zeichensatz ins RAM kopiert ist, können Werte direkt in die entsprechenden Adressen gePOKEt werden. Die Anfangsadresse eines Zeichens errechnet sich (wie schon beschrieben) aus

Anfang = 41984+8*ASCII (=40700+8*ASCII)

Um das erste Byte des Buchstabens 'A' (ASCII 65) mit dem Wert 255 zu laden, brauchen Sie nur

POKE 41984+8*65, 255 (POKE 40700+8*65, 255)

einzugeben. Das 'A' erscheint ab sofort mit einem aufgesetzten Balken (11111111 = 255). Mit

POKE 41984+8*65, 24 (POKE 40700+8*65, 24)

bringen Sie das 'A' wieder in seine ursprüngliche Form.

Das Programm „Zeichengenerator" arbeitet nach diesem Verfahren und bietet eine komfortable Lösung, Zeichen zu erstellen, zu verändern und abzuspeichern. Das sehr umfangreiche Programm verfügt über einige interessante Besonderheiten, wie z.B. das Drehen oder Invertieren eines Zeichens. Ein erstellter Zeichensatz kann abgespeichert und in ein beliebiges Programm zugeladen werden.
In der Bildschirmmitte befindet sich ein 8*8 Felder großes Raster, welches der Form eines Zeichens entspricht. Mit Hilfe der 4 Cursor-Tasten kann der Cursor innerhalb dieses Rasters hin- und herbewegt werden. Bei Betätigung der

COPY-Taste wird das Feld, auf dem sich der Cursor momentan befindet, entweder gesetzt oder aber gelöscht. Jede Veränderung wird auf der linken Bildschirmseite sofort als dezimaler Wert registriert. Das Zeichen mit dem ASCII-Wert 0 zeigt immer die aktuelle Form des bearbeiteten Zeichens in Originalgröße an.

H – Zeichen holen
Es wird ein vorhandenes Zeichen (ASCII 1-255) in das 8*8 Raster übergeben und kann bearbeitet werden.

A – Zeichen ablegen
Ein erstelltes oder bearbeitetes Zeichen wird einem ASCII-Code zugeordnet.

C – Zeichen löschen
Löscht das 8*8 Feld

I – Zeichen invertieren
Invertiert das 8*8 Feld. Ein gesetzter Punkt wird gelöscht, ein gelöschter gesetzt.

D – Zeichen drehen
Das vorhandene Zeichen wird um 90 Grad gegen den Uhrzeigersinn gedreht.

L – Zeichensatz laden
Ein schon erstellter, auf Kassette/Diskette abgelegter Zeichensatz wird eingelesen. Der sich momentan im Speicher befindliche Zeichensatz wird gelöscht.

S – Zeichensatz speichern
Der (neu) erstellte Zeichensatz wird auf der Kassette/Diskette abgelegt.

Q – Programmende.

```
1000 REM ************************************ M.W.Thoma **
1010 REM *                                               *
1020 REM *        Z E I C H E N - G E N E R A T O R      *
1030 REM *                                               *
1040 REM *************************************************
1050 DIM bit(8,8),help(8,8),byte(8)
1060 INK 0,0:INK 1,19:BORDER 0
1070 SPEED WRITE 1
1080 MODE 2
1090 SYMBOL AFTER 0
1100 start=HIMEM+1
```

```
1110 FOR i=0 TO 7
1120 POKE start+i,0
1130 NEXT i
1140 WINDOW#0,1,30,7,20
1150 WINDOW#1,34,50,7,23
1160 WINDOW#2,1,80,1,5
1170 WINDOW#3,54,80,7,25
1180 WINDOW#4,1,30,22,25
1190 LOCATE#2,25,4
1200 PRINT#2, "Z E I C H E N - G E N E R A T O R"
1210 LOCATE 24,7
1220 ORIGIN 268,40
1230 FOR y= 0 TO 256 STEP 32
1240 MOVE 0,y
1250 DRAW 128,y,1
1260 NEXT y
1270 FOR x=0 TO 16*8 STEP 16
1280 MOVE x,0
1290 DRAW x,256,1
1300 NEXT x
1310 LOCATE#3,1,3
1320 PRINT#3," <H> Zeichen holen"
1330 PRINT#3," <A> Zeichen ablegen"
1340 PRINT#3," <C> Zeichen loeschen"
1350 PRINT#3," <I> Zeichen invertieren"
1360 PRINT#3," <D> Zeichen drehen"
1370 PRINT#3," <L> Zeichensatz laden"
1380 PRINT#3," <S> Zeichensatz speichern"
1390 PRINT#3," <Q> Programmende"
1400 PRINT#3," <";CHR$(240);"> Cursor nach oben"
1410 PRINT#3," <";CHR$(241);"> Cursor nach unten"
1420 PRINT#3," <";CHR$(242);"> Cursor nach links"
1430 PRINT#3," <";CHR$(243);"> Cursor nach rechts"
1440 PRINT#3
1450 PRINT#3," <copy> Punkt setzen oder"
1460 PRINT#3,"        loeschen !"
1470 LOCATE 1,3
1480 PRINT "Geladenes Zeichen :"
1490 PRINT "ASCII-Code        :"
1500 FOR i=0 TO 7
1510 PRINT "Byte";i;"  :"
1520 NEXT i
1530 PRINT
1540 PRINT"Neues Zeichen   :"
1550 GOSUB 3430
1560 cz=16:cs=2
1570 '
1580 REM ***************************************************
1590 REM *                                                 *
1600 REM *                Tastaturabfrage                  *
1610 REM *                                                 *
1620 REM ***************************************************
1630 in$=INKEY$:IF in$="" THEN GOSUB 3650: GOTO 1630
1640 IF in$=CHR$(240) AND cz> 2 THEN GOSUB 3710:cz=cz-2
1650 IF in$=CHR$(241) AND cz<16 THEN GOSUB 3710:cz=cz+2
```

```
1660 IF in$=CHR$(242) AND cs> 2 THEN GOSUB 3710:cs=cs-2
1670 IF in$=CHR$(243) AND cs<16 THEN GOSUB 3710:cs=cs+2
1680 IF in$=CHR$(224) THEN GOSUB 3840:GOSUB 3920:GOSUB 3430
1690 IF UPPER$(in$)="H" THEN GOSUB 1790
1700 IF UPPER$(in$)="A" THEN GOSUB 2080
1710 IF UPPER$(in$)="C" THEN GOSUB 2240
1720 IF UPPER$(in$)="I" THEN GOSUB 2440
1730 IF UPPER$(in$)="D" THEN GOSUB 2600
1740 IF UPPER$(in$)="L" THEN GOSUB 2800
1750 IF UPPER$(in$)="S" THEN GOSUB 3070
1760 IF UPPER$(in$)="Q" THEN GOSUB 3260
1770 GOTO 1580
1780 '
1790 REM ***************************************************
1800 REM *                                                 *
1810 REM *                Zeichen holen                    *
1820 REM *                                                 *
1830 REM ***************************************************
1840 PAPER#4,1:PEN#4,0:CLS#4
1850 PRINT#4," <H> Zeichen holen !!!"
1860 LOCATE#4,2,2
1870 INPUT#4,"ASCII-Code:";ch
1880 IF ch<1 OR ch>255 THEN 2050
1890 FOR i=0 TO 7
1900 byte(i+1)=PEEK(start+ch*8+i)
1910 POKE start+i,byte(i+1)
1920 NEXT i
1930 FOR i= 1 TO 8
1940 FOR j=0 TO 7
1950 IF (byte(i) AND 2^j)=2^j THEN bit(8-j,i)=1:GOTO 1970
1960 bit(8-j,i)=0
1970 NEXT j
1980 NEXT i
1990 FOR i=1 TO 8
2000 FOR j=1 TO 8
2010 cz=2*i:cs=2*j:GOSUB 3710
2020 NEXT j
2030 NEXT i
2040 GOSUB 3430
2050 PAPER#4,0:PEN#1,1:CLS#4
2060 RETURN
2070 '
2080 REM ***************************************************
2090 REM *                                                 *
2100 REM *                Zeichen ablegen                  *
2110 REM *                                                 *
2120 REM ***************************************************
2130 PAPER#4,1:PEN#4,0:CLS#4
2140 PRINT#4," <A> Zeichen ablegen nach :"
2150 LOCATE#4,2,2
2160 INPUT#4,"ASCII-Code:";nch
2170 IF nch<1 OR nch>255 THEN 2210
2180 FOR i= 0 TO 7
2190 POKE start+8*nch+i,byte(i+1)
2200 NEXT i
```

```
2210 PAPER#4,0:PEN#1,1:CLS#4
2220 RETURN
2230 '
2240 REM ***************************************************
2250 REM *                                                 *
2260 REM *                Zeichen loeschen                 *
2270 REM *                                                 *
2280 REM ***************************************************
2290 PAPER#4,1:PEN#4,0:CLS#4
2300 PRINT#4," <C> Zeichen loeschen !!!"
2310 PRINT#4," Sind Sie sicher (j/n) ??"
2320 a$=INKEY$:IF a$="" THEN 2320
2330 IF UPPER$(a$)="J" THEN 2360
2340 IF UPPER$(a$)="N" THEN 2050
2350 GOTO 2320
2360 FOR i=1 TO 8
2370 FOR j=1 TO 8
2380 bit(i,j)=0
2390 NEXT j
2400 byte(i)=0:POKE start+i-1,0
2410 NEXT i
2420 GOTO 1990
2430 '
2440 REM ***************************************************
2450 REM *                                                 *
2460 REM *               Zeichen invertieren               *
2470 REM *                                                 *
2480 REM ***************************************************
2490 PAPER#4,1:PEN#4,0:CLS#4
2500 PRINT#4," <I> Zeichen invertieren"
2510 FOR i=1 TO 8
2520 FOR j=1 TO 8
2530 bit(i,j)=bit(i,j) XOR 1
2540 NEXT j
2550 byte(i)=byte(i) XOR 255
2560 POKE start+i-1,byte(i)
2570 NEXT i
2580 GOTO 1990
2590 '
2600 REM ***************************************************
2610 REM *                                                 *
2620 REM *            Zeichen um 90 Grad drehen            *
2630 REM *                                                 *
2640 REM ***************************************************
2650 PAPER#4,1:PEN#4,0:CLS#4
2660 PRINT#4," <D> Zeichen 90 Grad drehen"
2670 FOR i=1 TO 8
2680 FOR j=1 TO 8
2690 help(9-j,i)=bit(j,i)
2700 NEXT j
2710 NEXT i
2720 FOR x=1 TO 8
2730 FOR y=1 TO 8
2740 bit(y,x)=help(x,y)
2750 NEXT y
```

```
2760 cz=x*2:cs=y*2:GOSUB 3920
2770 NEXT x
2780 GOTO 1990
2790 '
2800 REM ***************************************************
2810 REM *                                                 *
2820 REM *                Zeichensatz laden                *
2830 REM *                                                 *
2840 REM ***************************************************
2850 PAPER#4,1:PEN#4,0:CLS#4
2860 PRINT#4," <L> Zeichensatz laden"
2870 PRINT#4," Achtung !"
2880 PRINT#4," Die Daten gehen verloren!"
2890 PRINT#4," Sind Sie sicher (j/n) ?"
2900 a$=INKEY$:IF a$="" THEN 2900
2910 IF UPPER$(a$)="N" THEN 3040
2920 IF UPPER$(a$)="J" THEN 2940
2930 GOTO 2900
2940 CLS#4
2950 PRINT#4," Bereiten Sie die Kassette vor";
2960 PRINT#4,"    und druecken eine Taste !"
2970 a$=INKEY$:IF a$="" THEN 2970
2980 CLS#4
2990 INPUT#4," Name :";n$
3000 IF LEN(n$)=0 OR LEN(n$)>8 THEN 3040
3010 WINDOW SWAP 0,4
3020 LOAD n$+".bin",start
3030 WINDOW SWAP 0,4
3040 PAPER#4,0:PEN#4,1:CLS#4
3050 RETURN
3060 '
3070 REM ***************************************************
3080 REM *                                                 *
3090 REM *                 Zeichen sichern                 *
3100 REM *                                                 *
3110 REM ***************************************************
3120 PAPER#4,1:PEN#4,0:CLS#4
3130 PRINT#4," <S> Zeichensatz sichern"
3140 PRINT#4," Bereiten Sie die Kassette vor";
3150 PRINT#4,"    und druecken eine Taste !"
3160 a$=INKEY$:IF a$="" THEN 3160
3170 CLS#4
3180 INPUT#4," Name :";n$
3190 IF LEN(n$)=0 OR LEN(n$)>8 THEN 3230
3200 WINDOW SWAP 0,4
3210 SAVE n$+".bin",b,start,2048
3220 WINDOW SWAP 0,4
3230 PAPER#4,0:PEN#4,1:CLS#4
3240 RETURN
3250 '
3260 REM ***************************************************
3270 REM *                                                 *
3280 REM *                 Programmende                    *
3290 REM *                                                 *
3300 REM ***************************************************
```

```
3310 PAPER#4,1:PEN#4,0:CLS#4
3320 PRINT#4," <Q> Programmende"
3330 PRINT#4," Achtung !"
3340 PRINT#4," Die Daten gehen verloren!"
3350 PRINT#4," Sind Sie sicher (j/n) ?"
3360 a$=INKEY$:IF a$="" THEN 3360
3370 IF UPPER$(a$)="N" THEN 3230
3380 IF UPPER$(a$)="J" THEN 3400
3390 GOTO 3360
3400 MODE 1
3410 END
3420 '
3430 REM ***************************************************
3440 REM *                                                 *
3450 REM *              Ausgabe der Byte-Werte             *
3460 REM *                                                 *
3470 REM ***************************************************
3480 LOCATE 23,3
3490 PRINT CHR$(1);CHR$(ch)
3500 LOCATE 21,4
3510 PRINT USING "###";ch
3520 FOR i= 1 TO 8
3530 LOCATE 14,4+i
3540 PRINT USING "###";byte(i)
3550 NEXT i
3560 LOCATE 23,14
3570 PRINT CHR$(1);CHR$(0)
3580 RETURN
3590 '
3600 REM ***************************************************
3610 REM *                                                 *
3620 REM *                  Cursor blinken                 *
3630 REM *                                                 *
3640 REM ***************************************************
3650 LOCATE#1,cs,cz
3660 PRINT#1,CHR$(32)
3670 LOCATE#1,cs,cz
3680 PRINT#1,CHR$(143)
3690 RETURN
3700 '
3710 REM ***************************************************
3720 REM *                                                 *
3730 REM *               Punkt setzen/loeschen             *
3740 REM *                                                 *
3750 REM ***************************************************
3760 IF bit(cs/2,cz/2)=1 THEN 3800
3770 LOCATE#1,cs,cz
3780 PRINT#1,CHR$(32)
3790 RETURN
3800 LOCATE#1,cs,cz
3810 PRINT#1,CHR$(143)
3820 RETURN
3830 '
3840 REM ***************************************************
3850 REM *
```

```
3860 REM *                Bit setzen/loeschen                *
3870 REM *                                                   *
3880 REM ***************************************************
3890 IF bit(cs/2,cz/2)=1 THEN bit(cs/2,cz/2)=0: RETURN
3900 bit(cs/2,cz/2)=1: RETURN
3910 '
3920 REM ***************************************************
3930 REM *                                                   *
3940 REM *              Umrechnung Bit-Byte                  *
3950 REM *                                                   *
3960 REM ***************************************************
3970 byte(cz/2)=0
3980 FOR i= 0 TO 7
3990 byte(cz/2)=byte(cz/2)+(2^i*bit(8-i,cz/2))
4000 NEXT i
4010 FOR i=0 TO 7
4020 POKE start+i,byte(i+1)
4030 NEXT i
4040 RETURN
4050 REM ***************************************************
```

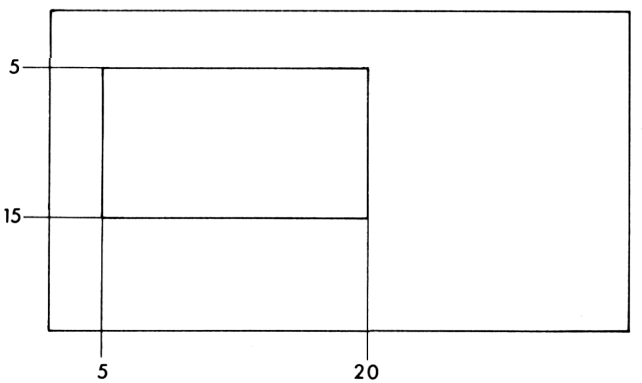
5.0 WINDOW - das Fenster im Bildschirm

Eine sehr mächtige Eigenschaft ist die sogenannte Window-Technik.

Windows oder Fenster sind im Bereich der Heim- und Homecomputer bis heute nahezu undenkbar gewesen. Auch hier hat der CPC-464 einen neuen Standard gesetzt. Windows (engl. Fenster) sind Bildschirmbereiche, die unabhängig voneinander bearbeitet werden können, ohne Einfluß auf den anderen Teil des Bildschirms zu nehmen. Mit bis zu 8 verschiedenen Fenstern kann gleichzeitig auf dem Bildschirm gearbeitet werden. Jedes der Fenster erhält eine laufende Nummer zwischen 0 und 7.
Alle Fenster haben im Einschaltzustand (Reset) dieselbe Größe und füllen jeweils den gesamten Bildschirm aus. Der eigentliche Bildschirm, also der, auf dem normalerweise gearbeitet wird, ist das Fenster 0. Die Größe der Fenster läßt sich sehr einfach mit dem Befehl

WINDOW #nr, sa, se, za, ze

bestimmen. 'nr' kennzeichnet die Fensternummer. Die folgenden Daten bestimmen die Größe des Fensters. 'sa' und 'se' geben die Werte für Spaltenanfang und Spaltenende, 'za' und 'ze' die Werte für Zeilenanfang und Zeilenende wieder. Als Beispiel soll ein Fenster zwischen Spalte 5 und 20 sowie zwischen Zeile 5 und 15 liegen.



Ordnen Sie dieses Fenster der Nummer 0 zu,

WINDOW #0, 5, 20, 5, 15

so ist der „Arbeitsbildschirm" auf die obige Größe festgelegt. Zu beachten ist, daß die Spaltenwerte in Abhängigkeit des jeweils gewählten Bildschirm-Modus (20, 40 oder 80 Zeichen) liegen müssen. Alle Ausgaben erscheinen ab sofort nur noch in diesem Bereich. Probieren Sie es einmal aus:

```
10 MODE 1
20 WINDOW #0, 5, 20, 5, 15
30 FOR i=1 TO 1000
40 PRINT i
50 NEXT i
60 END
```

Die Eigenschaften wie „Scrolling", Zeilenumbruch etc. bleiben auch in diesem Fenster erhalten, wobei auf den restlichen Bildschirm keinerlei Einfluß genommen wird. Sind mehrere Fenster definiert, muß weiterhin bestimmt werden, welchem Fenster eine Information zugeordnet sein soll. Um in das Fenster 1 „Hallo" zu PRINTen, muß

a) das Fenster definiert sein und
b) die PRINT-Anweisung mit der Fensternummer gekennzeichnet sein.

```
10 MODE 1
20 WINDOW #0, 1, 40, 1, 10
30 WINDOW #1, 1, 40, 11, 20
40 PRINT #1, "Ich schreibe im Fenster 1."
50 PRINT #0, "Ich schreibe im Fenster 0."
60 END
```

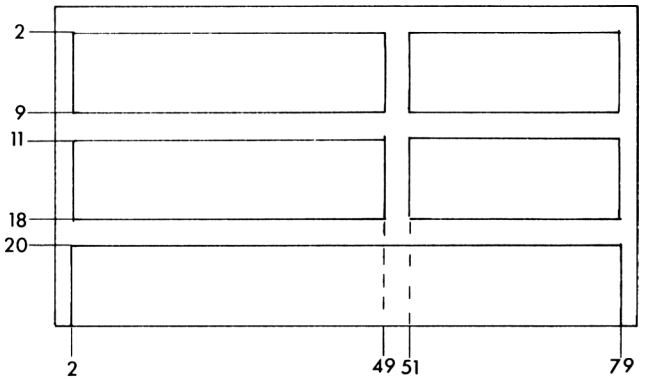
Alle Befehle, die eine Ausgabe auf dem Bildschirm bewirken (PRINT, INPUT, PEN, PAPER, LOCATE, CLS, LIST) müssen die Information beinhalten, auf welchen Bildschirm sie sich beziehen.

```
INPUT #2, ''Wert:'';I
CLS #5
LIST #6
```

Wird hinter den Befehlen keine Fensternummer angegeben, so interpretiert der CPC-464 es automatisch als Fenster 0.

```
CLS   = CLS #0
PRINT = PRINT #0
LIST  = LIST #0
```

Das Programm „5 Fenster" arbeitet mit 5 verschiedenen Fenstern, die (im Modus 2) an folgenden Stellen liegen:



Nach Eingabe der Fensternummer und eines Textes wird dieser Text in das jeweils bestimmte Fenster geschrieben (Programmende mit der „ESCAPE"-Taste).

```
1000 REM ************************************** M.W.Thoma **
1010 REM *                                                *
1020 REM *                    5 Fenster                   *
1030 REM *                                                *
1040 REM ***************************************************
1050 MODE 2
1060 WINDOW #0,2,79,20,24
1070 WINDOW #1,2,49,2,9
1080 WINDOW #2,51,79,2,9
1090 WINDOW #3,2,49,11,18
1100 WINDOW #4,51,79,11,18
1110 FOR i=0 TO 4
1120 PRINT #i,"Window #";i
1130 NEXT i
1140 INPUT "Fenster Nr.:";nr
1150 INPUT "Text       :";t$
1160 PRINT #nr,t$
1170 GOTO 1140
1180 REM ***************************************************
```

Jedes Fenster ist einem anderen Cursor zugeordnet, so daß der Computer sich immer merken kann, wo die letzte Ausgabe im Fenster stattgefunden hat. Das Positionieren der Cursor ist ebenfalls mit dem Befehl

LOCATE #nr, spalte, zeile

möglich.
Bei den INPUT-Anweisungen in den Programmzeilen 1140 und 1150 ist keine Fensternummer angegeben worden und sie werden somit automatisch dem Fenster 0 zugeordnet.

Alle Fenster lassen sich nicht nur räumlich sondern auch farblich anders gestalten. Je nach gewähltem Modus lassen sich die Fenster in 2, 4 oder 16 Farben gestalten. Die Hintergrund- und Zeichenfarbe der Fenster können unabhängig voneinander eingestellt werden. Mit

PAPER #nr, faßnummer
und PEN #nr, faßnummer

werden die Farben für die jeweiligen Fenster (nr) bestimmt. Im Programm „Farbige Fenster" wird von dieser Möglichkeit Gebrauch gemacht. In den Programmzeilen 1100 bis 1140 werden alle Fenster und der Border auf andere Farben gesetzt.

```
1000 REM ************************************ M.W.Thoma **
1010 REM *                                               *
1020 REM *                  Farbige Fenster              *
1030 REM *                                               *
1040 REM *************************************************
1050 MODE 1
1060 WINDOW #0,1,40,1,6
1070 WINDOW #1,1,40,7,12
1080 WINDOW #2,1,40,13,18
1090 WINDOW #3,1,40,19,25
1100 PAPER #0,0:PEN #0,1:CLS #0
1110 PAPER #1,1:PEN #1,0:CLS #1
1120 PAPER #2,2:PEN #2,3:CLS #2
1130 PAPER #3,3:PEN #3,2:CLS #3
1140 BORDER 2
1150 '
1160 PRINT #0,"Fenster Nr. 0  (Blau/Gelb)"
1170 PRINT #1,"Fenster Nr. 1  (Gelb/Blau)"
1180 PRINT #2,"Fenster Nr. 2  (Blaugruen/Hellrot)"
1190 PRINT #3,"Fenster Nr. 3  (Hellrot/Blaugruen)"
1200 END
1210 REM *************************************************
```

Interessant ist auch die Möglichkeit, die Zuordnung der Fensternummer zu vertauschen (eng. swap = tauschen). Durch den Befehl

WINDOW SWAP fenster1, fenster2

wird die Zuordnung der Fenster „fenster1" und „fenster2" vertauscht.

WINDOW SWAP 0, 1

bewirkt, daß alle Ausgaben, die normalerweise in Fenster 0 stattgefunden haben, nun im Fenster 1 stattfinden.
Fügen Sie im Programm die Zeile

```
1150 WINDOW SWAP 1, 2
```

ein und starten das Programm mit RUN. Auf dem Bildschirm erkennen Sie sofort, daß die Fenster 1 und 2 miteinander vertauscht worden sind. Probieren Sie auch einmal

```
1150 WINDOW SWAP 0, 3
```

aus. Die „READY"-Meldung erscheint nun im unteren Fenster, welches nach dem SWAP-Befehl nun Fenster 0 ist.

Die Möglichkeiten der Window-Technik sind eine der entscheidenden Leistungsmerkmale des CPC-464. Nach der Einrichtung der Fenster ist ihre Handhabung sehr einfach, da allen Befehlen nur die Fensternummer nachgestellt werden muß. Die einmal definierten Fenster bleiben solange bestehen, bis entweder ihr Ort verändert oder ein MODE-Befehl ausgeführt wird.

6.0 Hochauflösende Grafik

Unabhängig vom Zeichensatz ist es mit einfachen BASIC-Befehlen möglich, einzelne Bildschirmpunkte in den verschiedenen Farben zu setzen. Durch das programmierte Setzen von Bildschirmpunkten ist es relativ einfach, komplexe Grafiken zu erstellen. Der Grafik-Befehlssatz des CPC-464 ist recht sparsam ausgefallen. So sind außer dem Positionieren des Grafikcursors, Setzen eines Punktes in verschiedenen Farben und dem Zeichnen von Geraden keinerlei weitere Grafikfunktionen vorhanden. Besonders auffällig ist das Fehlen von Funktionen, wie z.B. zum Zeichnen einer Ellipse oder zum Zeichnen von Rechtecken, die man heute von einem Computer mit solch enormen Grafikfähigkeiten erwarten kann.

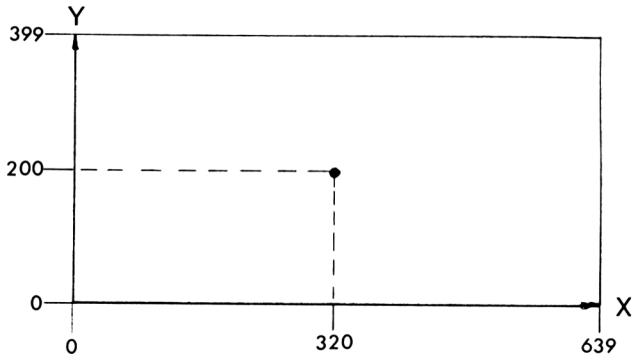
In diesem Kapitel wird der Bereich der hochauflösenden Grafik anhand vieler Programme und Unterprogramme erläutert. Die Unterprogramme sind so gestaltet worden, daß sie ohne weiteres in andere, bestehende Programme zugeladen werden können (siehe 19.4). Am Ende dieses Kapitels finden Sie zwei Funktions-Plot-Programme, die die fast professionellen Qualitäten des CPC-464 unter Beweis stellen.

6.1 ORIGIN - der Nullpunkt

Damit ein Punkt auf dem Bildschirm exakt lokalisiert werden kann, ist eine bestimmte Zuordnung notwendig. Der Bildschirm wird in ein gebräuchliches X/Y-Koordinatensystem eingeteilt. Die Waagerechte entspricht der X-Achse und die Senkrechte der Y-Achse.

Die X-Achse ist in 640 Punkte und damit in seine maximale Bildpunktanzahl geteilt. Die Y-Achse, die aus 200 Bildpunkten besteht, ist allerdings in 400 Einheiten gegliedert worden. Das hat zur Folge, daß zwei Y-Achsenwerte jeweils demselben Bildpunkt entsprechen. Der Nullpunkt beider Achsen befindet sich normalerweise in der unteren, linken Ecke.

Mit einer X/Y-Definition ist jeder der 128000 möglichen Bildpunkte genau zu lokalisieren.



Im obigen Beispiel wurde der Punkt

X=320 / Y=200

bestimmt. Dieser Punkt befindet sich in der Mitte des Bildschirms. Der Punkt in der unteren, linken Ecke läßt sich entsprechend mit

X=0 / Y=0 oder
X=0 / Y=1 (!!)

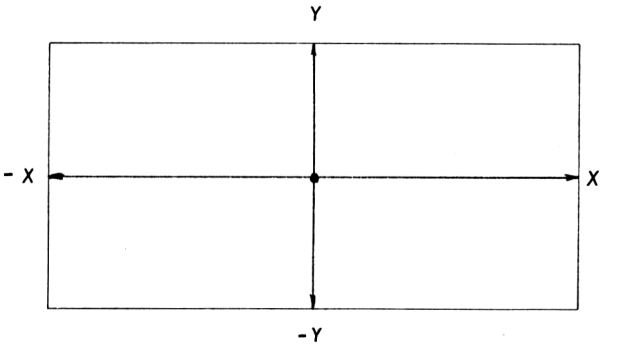
bestimmen. Daß der Y-Wert auch '1' sein kann, liegt an der Doppelbelegung der Y-Achse!!
Der Ursprung (Nullpunkt der Achsen) läßt sich wahlfrei auf jeden beliebigen Punkt festlegen. Im Einschaltzustand (reset) befindet er sich in den Koordinaten X=0/Y=0. Der Befehl ORIGIN (Herkunft, Ursprung) legt durch eine X/Y-Angabe den Ursprung auf einen neuen Punkt.

ORIGIN x-achse, y-achse

Damit der Ursprung in der Mitte des Bildschirms liegt, sind die Werte

ORIGIN 320, 200

anzugeben. Der ORIGIN-Befehl bezieht sich immer auf das „normale" Koordinatensystem (X=0-639 / Y=0-399), auch wenn der Ursprung verlegt worden ist. Nun sieht das (gedachte) Koordinatensysten folgendermaßen aus:



Es ist ein neues Koordinatensystem und damit auch eine andere Bildpunktzuordnung entstanden. Die Koordinaten 0/0 befinden sich genau in der Mitte des Bildschirms. Der Punkt in der unteren, linken Ecke hingegen bei

X= -320 und Y= -200

Die X-Achse verläuft von

−320 nach +319 (640 Punkte)

und die Y-Achse von

−200 nach +199 (400 Einheiten/200 Punkte)

Es ist auch möglich, den Ursprung des Koordinatensystems außerhalb des Bildschirms zu legen.

ORIGIN -100, 0

Damit liegt die untere, linke Ecke bei den Koordinaten X= +100 und Y=0! Es ist aber in der Regel nicht notwendig, den Ursprung außerhalb des Bildschirms zu legen und sollte aus Gründen der Übersichtlichkeit stets vermieden werden.

6.2 PLOT - der Punkt

Mit dem Befehl

PLOT x, y, faßnummer

wird ein Punkt absolut an den Ort, der durch die Werte für die X- und Y-Koordinate bestimmt ist, gesetzt. Die „faßnummer" gibt die Farbe des zu setzenden Punktes an (siehe 2.0). Durch

MODE 2: PLOT 0 ,0, 1

wird der Punkt in der unteren, linken Ecke (bzw. an den aktuellen Koordinaten-Nullpunkt in der Farbe aus Faß 1 (Zeichenfarbe) gesetzt. „Gelöscht" wird ein Punkt, indem er in der aktuellen Hintergrundfarbe (PAPER) gesetzt wird und damit nicht mehr sichtbar ist.

PLOT 0, 0, 0

Ein senkrechter Strich entsteht durch das Setzen mehrerer Punkte in der Y-Achse:

```
10 MODE 2
20 FOR y=0 TO 399 STEP 2
30 PLOT 0, y, 1
40 NEXT y
50 END
```

Zu beachten ist der Parameter „STEP 2" in der Programmzeile 20. Da jeweils zwei Y-Werte denselben Punkt kennzeichnen (0/1, 2/3, 4/5, ..), ist es nicht sinnvoll, jeden Y-Wert zu setzen. Das Ergebnis wäre zwar identisch, allerdings bei einer doppelt so langen Verarbeitungszeit.

Die Auflösung der X-Achse ist bekanntlich abhängig vom gewählten Bildschirm-Modus (siehe 3.2). Je nach Modus werden 1, 2 oder 4 Punkte in der X-Achse gleichzeitig in der gewählten Farbe gesetzt. Um im Modus 0 (20 Zeichen) eine waagerechte Linie zu zeichnen, muß nur jeder 4. Punkt der X-Achse gesetzt werden.

```
10 MODE 0
20 FOR x=0 TO 639 STEP 4
30 PLOT x, 0, 1
40 NEXT x
50 END
```

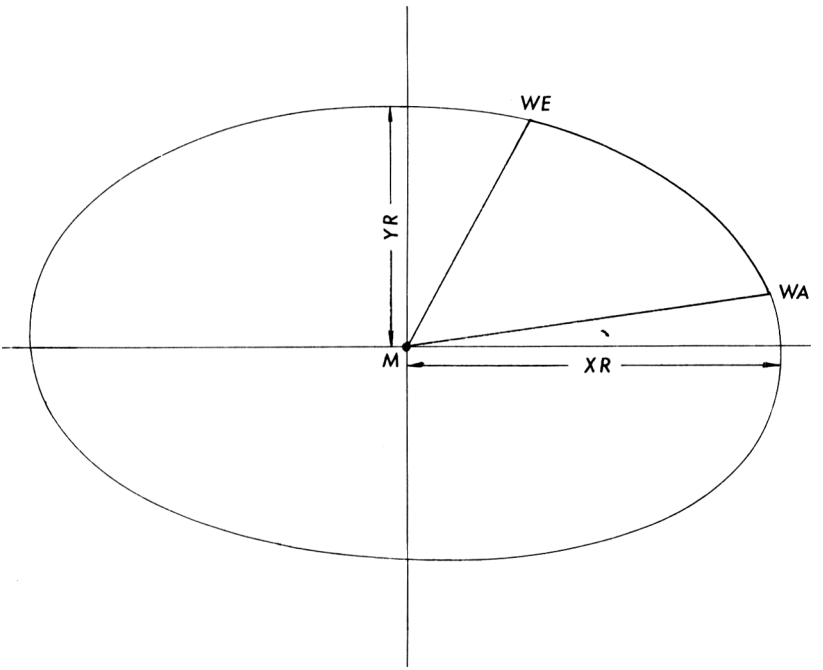
Im Modus 1 (40 Zeichen) muß entsprechend jeder 2. und im Modus 2 jeder Punkt gesetzt werden, um eine geschlossene Linie zu erzielen.

6.3 Die Ellipse

Als erste praktische Anwendung des PLOT-Befehls soll das Zeichnen einer Ellipse bzw. eines Ellipsenteiles realisiert werden.

Der Kreis ist eine Sonderform der Ellipse: Der X-Radius ist gleich dem Y-Radius (YR=XR). Bei einer Ellipse ist der X- und Y-Radius unterschiedlich. Der Ort, wo sich eine Ellipse im System befindet, wird durch seinen Mittelpunkt (XM/YM) bestimmt. Die X- und Y-Koordinaten für einen Ellipsenpunkt berechnen sich aus einer Sinus- und einer Cosinus-Funktion.

X=XR*COS(Winkel)
Y=YR*SIN(Winkel)



Der Winkel gibt die Gradzahl (0-360 Grad) des zu berechnenden Punktes an. Um eine geschlossene Ellipse zu erreichen, muß der Winkel von 0 bis 360 Grad berechnet werden. Der Ort einer Ellipse ist durch XM;YM bestimmt und wird durch

X=XM+XR*COS(Winkel)
Y=YM+YR*SIN(Winkel)

berücksichtigt. Ein Ellipsen bzw. Kreisteil (z. B. von 45 Grad nach 135 Grad) wird erzielt, indem nur diese Werte errechnet und als Punkte dargestellt werden.

```
60000 REM ************************************ M.W.Thoma **
60010 REM *          Ellipse  und Ellipsenteile           *
60020 REM *                                               *
60030 REM *      wa = Winkelanfang ( < we )               *
60040 REM *      we = Winkelende   ( > wa )               *
60050 REM *      xm = X-Koordinate des Mittelpunktes      *
60060 REM *      ym = Y-Koordinate des Mittelpunktes      *
60070 REM *      xr = X-Radius                            *
60080 REM *      yr = Y-Radius                            *
60090 REM *       f = Zeichenfarbe                        *
60100 REM *                                               *
60110 REM *************************************************
60120 DEG
60130 FOR winkel = wa TO we
60140 x=xm+xr*COS(winkel)
60150 y=ym+yr*SIN(winkel)
60160 PLOT x,y,f
60170 NEXT winkel
60180 RETURN
60190 REM *************************************************
```

Als Übergabeparameter sind die bekannten Variablen benutzt worden (siehe Remark-Zeilen). Vergessen Sie bitte nicht die Farbe zu setzen (f=1), da sonst die Punkte in der Hintergrundfarbe gesetzt werden (f=0) !!

Probieren Sie das Unterprogramm einmal aus:

```
10 MODE 2
20 ym=200:xr=75:yr=200:wa=0:we=360:f=1
30 FOR xm=220 TO 420 STEP 100
40 GOSUB 60000
50 NEXT xm
60 ym=200:xm=320:xr=200:yr=100:wa=0:we=360:f=1:GOSUB
   60000
70 END
```

6.4 Der Grafik-Cursor

Immer dann, wenn ein Grafik-Befehl ausgeführt wird, zeigt ein nicht sichtbarer „Grafikcursor" auf die aktuelle X/Y-Position. Nach einem PLOT-Befehl z.B. zeigt der Grafikcursor auf den gesetzten Punkt. Diese Position bleibt solange erhalten, bis ein neuer Grafik-Befehl ausgeführt wird. Nach dem Einschalten (reset) und nach einem ORIGIN-Befehl zeigt der Grafikcursor auf den Ursprung, also X=0;Y=0. Die aktuelle Grafikcursorposition läßt sich mit den Befehlen

```
XPOS und YPOS
```

abfragen, wobei XPOS die X- und YPOS die Y-Koordinate wiedergibt.

```
PRINT XPOS, YPOS
  0      0
```

Ein weiterer „Punkt-Setz"-Befehl, der einen Punkt relativ zur aktuellen Grafikcursorposition setzt, ist im BASIC des CPC-464 enthalten:

```
PLOTR x, y, faßnummer
```

Hier gibt „x" und „y" den Versatz in die jeweilige Richtung von der aktuellen Grafikcursorposition an, wobei der Punkt in der Farbe aus „faßnummer" gesetzt werden soll. Ein Beispiel:

```
PLOTR 10, 0, 1
```

Der Punkt wird 10 Punkte in X-Richtung und 0 Punkte in Y-Richtung, ausgehend von der momentanen Grafikcursorposition, gesetzt. Die Cursor-Position ist dann wieder die neue absolute Position.

6.5 MOVE - Positieren des Grafik-Cursors

Mit Hilfe des Befehls

```
MOVE x, y
```

kann der Grafikcursor auf einen Punkt innerhalb des Koordinatensystems positioniert werden (eng. move = bewegen). Nach einem MOVE-Befehl zeigt der Grafikcursor auf die neuen, durch MOVE bestimmten Koordinaten.

```
10 MODE 2
20 MOVE 100,200
30 PRINT XPOS, YPOS
40 END
```

Wie beim PLOT-Befehl ist auch beim MOVE-Befehl ein „relatives" Setzen des Grafikcursors möglich. Der Befehl

MOVER x, y

gibt wiederrum den Versatz in X- und Y-Richtung an. Zur Verdeutlichung ein kleines Beispiel:

```
10 MOVE 100, 100
20 MOVER 10, 30
30 PRINT XPOS, YPOS
```

In der Zeile 10 wurde der Grafikcursor auf die Position X=100 und Y=100 im aktuellen Koordinatensystem gesetzt. Er zeigt jetzt auf 100;100. In Zeile 20 ist nun der Grafikcursor von der absoluten Position um 10 X- und 30 Y-Einheiten versetzt worden, so daß die neue, absolute Position X=110 und Y=130 ist.

6.6 DRAW - Ziehen einer Linie

Der Draw-Befehl (eng. draw = ziehen, zeichnen) ist der einzige komplexe Zeichenbefehl innerhalb des Schneider-BASICs. Alle Formen, Körper und Zeichnungen, die Sie mit dem CPC-464 erstellen wollen, müssen also mit dem PLOT und dem DRAW-Befehl verwirklicht werden (was auch ohne weiteres möglich ist). Der Befehl

DRAW x, y, faßnummer

gibt den Ort an, von dem aus eine Gerade zur aktuellen Grafikcursorposition gezogen werden soll. Befindet sich der Grafikcursor z.B. an dem Ort X=0 und Y=0 und Sie geben den Befehl

```
DRAW 100, 0, 1
```

ein, so entsteht eine Gerade zwischen den X-Koordinaten 0 und 100. Die Handhabung des DRAW-Befehls ist in der Praxis weitestgehend unproblematisch. Das einzige Problem besteht darin, daß Sie immer wissen müssen, wo sich der Grafikcursor z. Z. befindet, denn er bildet den Endpunkt der zu zeichnenden Linie. Nach dem DRAW-Befehl zeigt der Grafikcursor wiederum auf die im Befehl angegebenen Koordinaten.

Durch einfache Schleifen lassen sich schon recht nette Grafiken erstellen. „Spielen" Sie ruhig ein bißchen mit dem Befehl, um so das Verständnis des DRAW-Befehls zu erlangen.

Natürlich darf auch beim Zeichnen einer Geraden der relative Zeichne-Befehl nicht fehlen:

```
DRAWR x, y, faßnummer
```

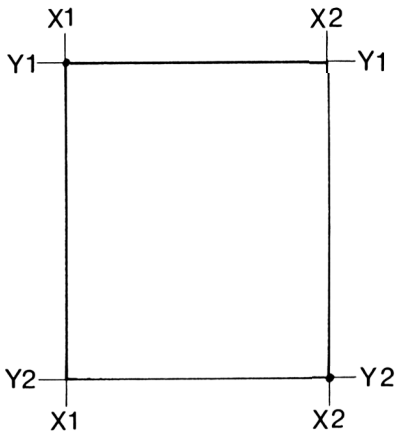
„x" und „y" geben wieder den Versatz von der aktuellen Grafikcursorposition an. Das kleine Demonstrationsprogramm zeichnet eine Treppe mit DRAWR-Befehlen:

```
10 MODE 2
20 MOVE 0,0
30 FOR z=1 TO 40
40 DRAWR 10,0,1
50 DRAWR 0,10,1
60 NEXT z
70 END
```

6.7 Rechtecke

Nachdem mit dem Programm zum Zeichnen einer Ellipse die Unterprogrammbibliothek begonnen wurde, folgen nun 2 verschiedene Möglichkeiten zum Zeichnen eines Rechtecks.

Der Ort und die Größe eines Rechteckes sollen mit den Angaben der oberen, linken und unteren, rechten Ecke bestimmt werden. Alle anderen Eckpunkte des Rechtecks ergeben sich wie aus der Zeichnung ersichtlich:



Die Eckpunkte X1;Y1 (oben links) und X2;Y2 (unten rechts) des Rechtecks sowie die Zeichenfarbe (f) werden dem Unterprogramm übergeben.

```
61000 REM ************************************ M.W.Thoma **
61010 REM *                  Rechtecke  (1)               *
61020 REM *                                               *
61030 REM *          x1 = X-Koordinate oben               *
61040 REM *          y1 = Y-Koordinate oben               *
61050 REM *          x2 = X-Koordinate unten              *
61060 REM *          y2 = Y-Koordinate unten              *
61070 REM *           f = Zeichenfarbe                    *
61080 REM *                                               *
61090 REM *************************************************
61100 MOVE x1,y1
61110 DRAW x2,y1,f
61120 DRAW x2,y2,f
61130 DRAW x1,y2,f
61140 DRAW x1,y1,f
61150 RETURN
61160 REM *************************************************
```

Das zweite Programm zum Zeichnen eines Rechtecks ist speziell für den Modus 1 ausgelegt und füllt das Rechteck mit einer Farbe aus. Dabei ist das Füllen des Rechtecks so gestaltet, daß die Ränder nicht wieder übermalt werden. Im ersten Teil des Programms finden Sie die normale Rechteckroutine vom vor-

herigen Programm wieder (bis 61150). Das Ausfüllen des Rechtecks findet in den Zeilen 6118 0 bis 61210 statt. Es werden einfach Linien in der Füllfarbe (f2) in das Rechteck gezeichnet. Dabei muß beachtet werden, daß die schon gezeichneten Rechteckränder nicht wieder übermalt werden (sieht so besser aus)! Erzielt wird dieses, indem die Linie einen Bildschirmpunkt neben dem linken Rand beginnt (Zeile 61160) und einen Bildschirmpunkt vor dem rechten Rand endet (Zeile 61200). Das Addieren oder Subtrahieren mit Wert 2 resultiert aus dem gewählten Bildschirmmodus (MODE 1). Hier werden ja immer 2 Punkte gleichzeitig in der Zeichenfarbe gesetzt.

In den Zeilen 61160 und 61170 wird sichergestellt, daß X1 und Y1 die obere Ecke repräsentieren. Sie werden gegebenfalls getauscht, damit X2 größer als X1 und Y1 größer als Y2 ist.

```
61000 REM ************************************ M.W.Thoma **
61010 REM *          Rechtecke  (2)    ( MODE 1 )          *
61020 REM *                                                 *
61030 REM *           x1 = X-Koordinate oben                *
61040 REM *           y1 = Y-Koordinate oben                *
61050 REM *           x2 = X-Koordinate unten               *
61060 REM *           y2 = Y-Koordinate unten               *
61070 REM *           f1 = Zeichenfarbe                     *
61080 REM *           f2 = Fuellfarbe                       *
61090 REM *                                                 *
61100 REM ***************************************************
61110 MOVE x1,y1
61120 DRAW x2,y1,f1
61130 DRAW x2,y2,f1
61140 DRAW x1,y2,f1
61150 DRAW x1,y1,f1
61160 IF y1>y2 THEN y0=y1:y1=y2:y2=y0
61170 IF x2<x1 THEN x0=x1:x1=x2:x2=x0
61180 FOR i= y1+2 TO y2-2 STEP 2
61190 MOVE x1+2,i
61200 DRAW x2-2,i,f2
61210 NEXT i
61220 RETURN
61230 REM ***************************************************
```

Testen Sie dieses Unterprogramm mit dem folgenden kleinen Programm.

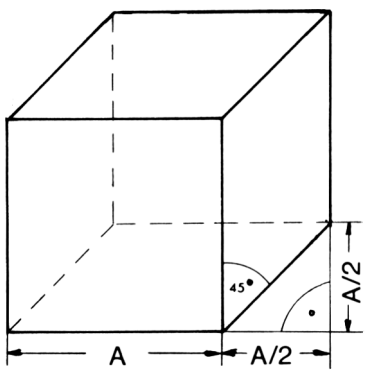
```
100 MODE 1
110 INK 0,15: BORDER 15
120 f1=2
130 FOR f2=1 TO 0 STEP -1
140 FOR z=1 TO 10
150 x1=z*50
```

```
160 y1=399
170 x2=(z+1)*50
180 y2=0
190 GOSUB 61000
200 NEXT z
210 NEXT f2
220 END
```

6.8 Quader - ein Körper

Wesentlich komplizierter als das Zeichnen einer Fläche ist das Zeichnen eines Körpers, vor allem dann, wenn die Größe und der Standort des Körpers variabel gehalten werden soll. Das Darstellen von Körpern auf einer 2-dimensionalen Fläche (Bildschirm) ist auf Grund der benötigten mathematischen Kenntnisse ein großes Problem und für viele Freizeitprogrammierer bis heute „schleierhaft" geblieben. Anhand eines Programms zum Zeichnen eines Quaders soll dieses Geheimnis ein wenig „gelüftet" und anschließend mit dem Programm zur Darstellung von 3-dimensionalen Funktionen (siehe 7.5, 10.0) vertieft werden.

Die bekannteste und auch einfachste Form der 3-dimensionalen Darstellung ist die sogenannte „Kavalierperspektive". Bei der „Kavalierperspektive" wird davon ausgegangen, daß die räumlichen Flächen in 45 Grad zur Waagerechten dagestellt werden. Damit ein räumlicher Effekt zustande kommt, werden die Flächen verkürzt dargestellt. Sehen Sie sich den Würfel einmal genauer an:

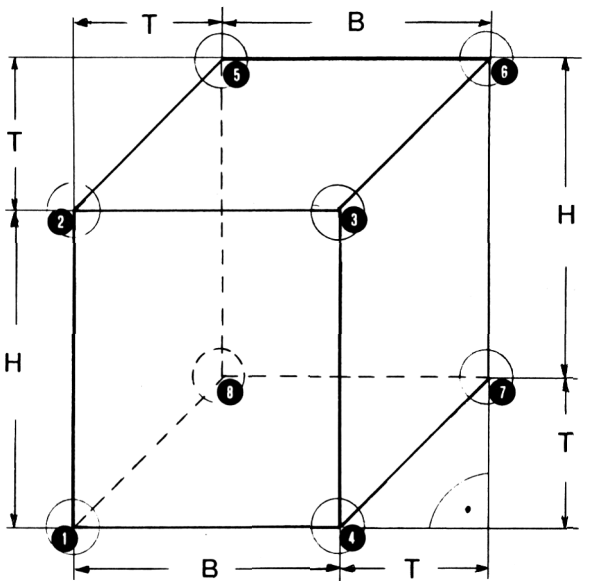


Bei einem Würfel besitzen alle Kanten die gleiche Länge (a) und bei einem Winkel von 45 Grad verkürzen sich die räumlichen Kanten genau um die Hälfte (a/2). Bildet man daraus ein rechtwinkliges Dreieck, so ergibt sich eine „räumliche Tiefe" von ebenfalls a/2.

Mit diesen Informationen ist es möglich, einen beliebig großen Quader an jedem Ort des Bildschirms zu zeichnen. Die Größe und der Standort des Quaders sollen durch folgende Werte bestimmt werden:

X1;Y1 = Standort unten links
h = Höhe des Quaders
b = Breite des Quaders
t = Tiefe des Quaders

Mit diesen Werten lassen sich alle Eckpunkte eines Quaders berechnen. Anhand der Zeichnung sind die Zusammenhänge leicht zu erkennen:



Alle Eckpunkte (p1-p8) besitzen dann folgende Werte:

p1 : x=x1	y=y1	(gegebene Werte)
p2 : x=x1	y=y1+h	
p3 : x=x1+b	y=y1+h	
p4 : x=x1+b	y=y1	
p5 : x=x1+t	y=y1+h+t	
p6 : x=x1+b+t	y=y1+h+t	
p7 : x=x1+b+t	y=y1+t	
p8 : x=x1+1	y=y1+t	(nicht sichtbar)

Diese Eckpunkte müssen nun nur noch durch den DRAW-Befehl miteinander verbunden werden. Das Unterprogramm „Quader" zeichnet (wie beschrieben) die sichtbaren Kanten des Quaders in den Programmzeilen 62130 bis 62240. Der nachfolgende Programmteil füllt ihn auf Wunsch (tr=1) mit einer Farbe. Den „Deckel" in der Farbe aus Faß 1 (62260-62310) und alle anderen Flächen mit der Farbe aus Faß 3 (62320-62360).

```
62000 REM ************************************ M.W.Thoma **
62010 REM *                                               *
62020 REM *    Zeichnen von (ausgefuellten) Quadern       *
62030 REM *                                               *
62040 REM * X1   : X-Koordinate Quader unten              *
62050 REM * Y1   : Y-Koordinate Quader links               *
62060 REM * H    : Hoehe des Quaders (0-200 Bildpunkte)   *
62070 REM * B    : Breite des Quaders (Bildpunkte)        *
62080 REM * T    : Tiefe des Quaders (Bildpunkte)         *
62090 REM * TR   : 0 = Quader transparent darstellen      *
62100 REM *        1 = Quader farbig ausfuellen           *
62110 REM *                                               *
62120 REM *************************************************
62130 MOVE X1,Y1
62140 DRAW X1,Y1+H,2
62150 DRAW X1+B,Y1+H,2
62160 DRAW X1+B,Y1,2
62170 DRAW X1,Y1,2
62180 MOVE X1,Y1+H
62190 DRAW X1+T,Y1+H+T,2
62200 DRAW X1+B+T,Y1+H+T,2
62210 DRAW X1+B,Y1+H,2
62220 MOVE X1+B,Y1
62230 DRAW X1+B+T,Y1+T,2
62240 DRAW X1+B+T,Y1+H+T,2
62250 IF tr=0 THEN 62360
62260 FOR S=Y1+2 TO Y1+H-2 STEP 2
62270 MOVE X1+2,S
62280 DRAW X1+B-2,S,1
62290 MOVER 4,2
```

```
62300 DRAW X1+B-2+T,S+T-2,1
62310 NEXT S
62320 FOR S=X1+4 TO X1+B STEP 2
62330 MOVE S,Y1+H+2
62340 DRAW S+T-4,Y1+H+T-2,3
62350 NEXT S
62360 RETURN
62370 REM ************************************************
```

Auch das Quader-Programm soll getestet werden:

```
100 MODE 1
110 tr=1:y1=0:b=50:t=25
120 FOR x1=0 TO 500 STEP 100
130 h=INT(RND (1)*200)
140 GOSUB 62000
150 NEXT x1
160 END
```

7.0 Grafische Darstellung von Werten

Das Hauptanwendungsgebiet der Computergrafik ist neben der Animation und den Spielprogrammen vor allem die Darstellung von Größen und Größenverhältnissen. Eine Verkaufsstatistik ist grafisch wesentlich leichter zu überschauen als eine lange Zahlenkolonne. Bei der grafischen Darstellung von Werten kommt es nicht so sehr auf die Genauigkeit der dargestellten Werte an, sondern mehr auf die Übersichtlichkeit: Tendenzen und Wertspitzen fallen sofort ins Auge.

Es gibt verschiedene Möglichkeiten, Werte grafisch darzustellen. Jede Darstellungsart hat ihre Vorzüge, die vor allem von der Anzahl der darzustellenden Werte abhängig ist. Auf den nächsten Seiten werden die gebräuchlichsten Arten der grafischen Darstellung von Werten beschrieben. Sie finden Programme zum/zur

* Balkendiagramm
* Werte-Plot
* Tortengrafik
* 3-dimensionalen Werte-Darstellung (Balken)

Alle Programme sind wieder als Unterprogramme ausgelegt und lassen sich ohne weiteres in bestehende Programme einbinden. Die Übergabevariablen sind in allen Programmen einheitlich, so daß Werte ohne große Schwierigkeiten mit allen Unterprogrammen dargestellt werden können.

7.1 Balkendiagramm

Als erste und wohl auch bekannteste Form der Wertedarstellung wird das Balkendiagramm beschrieben. Beim Balkendiagramm werden alle Werte als senkrechte oder waagerechte Linien dargestellt. Meistens sind die Werte vorher in ihrem prozentualen Anteil zum Maximalwert oder zur Summe umge-

rechnet, damit eine bessere Zuordnung der Größen möglich ist. Beim Programm „Balkendiagramm" sind alle Werte im Verhältnis zum größten (maximalen) Wert dargestellt. Der Maximalwert wird in der Schleife 50350 bis 50370 ermittelt und der Variablen „maximal" zugewiesen. Das Hauptproblem liegt im Anpassen der Balken an das Bildschirmkoordinatensystem, damit auch der größte Balken (Wert) noch voll auf dem Bildschirm sichtbar ist. Der Maximalwert muß ins Verhältnis zu der Verfügung stehenden Punktanzahl gesetzt werden. Im Programm stehen für die Darstellung der Balken 300 Y-Werte zur Verfügung.

Faktor = 300/maximal
Faktor = 300/600
Faktor = 0.5

Bei einem Maximalwert von z.B. 600 wird mit der obenstehenden Gleichung ein Faktor von 0.5 errechnet. Jeder Wert muß dann mit dem Faktor multipliziert werden, um an das Bildschirmsystem angepaßt zu werden. Z.B.

Y-Größe = 400*Faktor
Y-Größe = 400*0.5
Y-Größe = 200

Durch Multiplizieren des Maximalwerts (600) mit dem Faktor

600*0.5 = 300

wird wieder die maximale Y-Größe erzielt.
Eine zweite Anpassung ist in der X-Ebene notwendig. Damit alle Werte auf dem Bildschirm Platz finden, muß dafür gesorgt werden, daß die Breite aller Balken nicht größer als der zur Verfügung stehende Platz wird. Sind z.B. 550

X-Punkte vorhanden und es sollen 10 Balken dargestellt werden, so darf jeder Balken nur

$$550/10 = 55$$

X-Punkte breit sein.
Im Programm finden Sie die beiden Gleichungen in den Zeilen 50180 und 50440 wieder. Damit nicht alle Balken direkt aneinander gezeichnet werden und dadurch die Übersichtlichkeit verlorengeht, wird zwischen den Balken ein Abstand von 10% der Balkenbreite berücksichtigt (Zeile 50450).

Die Werte, die als Balken dargestellt werden sollen, befinden sich in dem Array „wert(...)". Wieviele Werte vorhanden sind, bestimmt die Variable „anzahl". Auch beim „Balkendiagramm" besteht wieder die Möglichkeit, die Balken mit einer Farbe auszufüllen (tr=1). Negative Werte sollen (und können) mit diesem Programm nicht dargestellt werden. Die Zeile 50470 prüft dieses noch einmal und übergeht gegebenenfalls solche Werte.

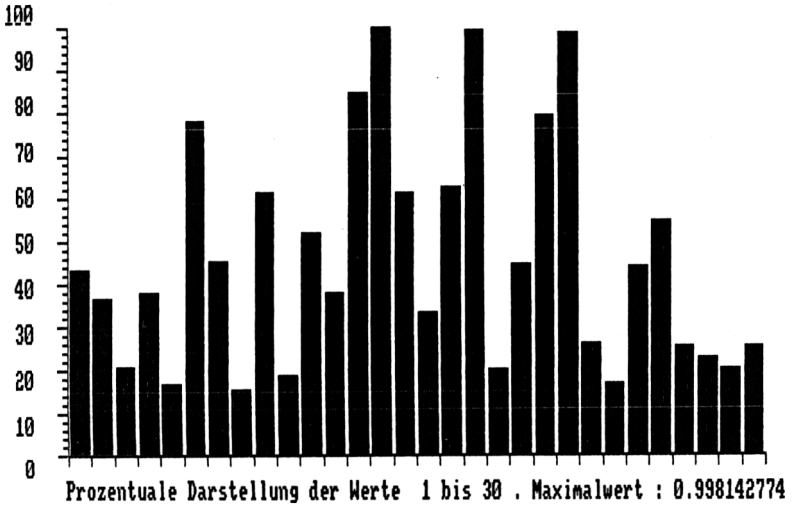
```
50000 REM ************************************ M.W.Thoma **
50010 REM *                Balkendiagramm                  *
50020 REM *                                                *
50030 REM *   wert (...) : Wertarray                       *
50040 REM *   anzahl     : Anzahl der vorhandenen Werte    *
50050 REM *   tr         : 0 = Balken transparent          *
50060 REM *              : 1 = Balken ausgefuellt          *
50070 REM *                                                *
50080 REM ******************************************************
50090 MODE 2
50100 ORIGIN 50,50
50110 MOVE 550,0
50120 DRAW 0,0,1:DRAW 0,300,1
50130 FOR i=0 TO 100 STEP 2
50140 MOVE 0,i*3
50150 IF (i/10)=INT(i/10) THEN DRAW -8,i*3,1
50160 DRAW -4,i*3,1
50170 NEXT i
50180 ver=550/anzahl
50190 FOR i=0 TO anzahl
50200 MOVE i*ver,0
50210 DRAW i*ver,-8,1
50220 NEXT i
50230 LOCATE 1,3
50240 FOR i=100 TO 0 STEP -10
```

```
50250 PRINT USING "###";i
50260 PRINT
50270 NEXT i
50280 '
50290 REM ************************************************
50300 REM *                                              *
50310 REM *              Groessten Wert suchen           *
50320 REM *                                              *
50330 REM ************************************************
50340 maximal=wert(1)
50350 FOR i=2 TO anzahl
50360 IF wert(i) > maximal THEN maximal=wert(i)
50370 NEXT i
50380 '
50390 REM ************************************************
50400 REM *                                              *
50410 REM *              Zeichnen des Diagramms          *
50420 REM *                                              *
50430 REM ************************************************
50440 faktor=300/maximal
50450 abstand=ver*0.1
50460 FOR i=1 TO anzahl
50470 IF wert(i)<0 THEN PRINT CHR$(7);:GOTO 50570
50480 MOVE (i-1)*ver+abstand,0
50490 DRAW (i-1)*ver+abstand,wert(i)*faktor,1
50500 DRAW i*ver-abstand,wert(i)*faktor,1
50510 DRAW i*ver-abstand,0,1
50520 IF tr = 0 THEN 50570
50530 FOR j=1 TO wert(i)*faktor STEP 2
50540 MOVE (i-1)*ver+abstand,j
50550 DRAW i*ver-abstand,j,1
50560 NEXT j
50570 NEXT i
50580 LOCATE 7,24
50590 PRINT "Prozentuale Darstellung der Werte  1 bis";
50600 PRINT anzahl;". Maximalwert :";maximal
50610 RETURN
50620 REM ************************************************
```

Auch zum „Balkendiagramm“ ein kleines Testprogramm:

```
100 DIM wert(30)
110 anzahl=30
120 tr=1
130 '
140 FOR z=1 TO anzahl
150 wert(z)=RND(1)
160 NEXT z
170 GOSUB 50000
180 END
```

Das Ergebnis des Beispielprogramms könnte so aussehen (DIN A5)



7.2 Werte-Plot

Das zweite Unterprogramm zur grafischen Wertedarstellung ist der „Werte-Plot". Beim „Werte-Plot" wird die Größe der Werte als Punkte im Koordinatensystem gekennzeichnet, die anschließend mit einer Linie (Geraden) verbunden werden. Die Anpassung an das Bildschirmsystem ist vollständig identisch mit dem des „Balkendiagramms". Anstatt eines Punktes wird zur Kennzeichnung des Wertes ein kleines Rechteck gezeichnet (51470-51520). Mit Hilfe der Variablen „binden" können Sie bestimmen, ob die Punkte durch eine Linie miteinander verbunden werden sollen.

```
51000 REM ************************************ M.W.Thoma **
51010 REM *                  Werte-Plot                   *
51020 REM *                                               *
51030 REM *   wert(...)     : Wertearray                  *
51040 REM *   anzahl        : Anzahl der Werte            *
51050 REM *   binden        : 0 = Punkte nicht verbinden  *
51060 REM *                   1 = Punkte verbinden        *
51070 REM *                                               *
51080 REM *****************************************************
51090 MODE 2
51100 ORIGIN 50,50
51110 MOVE 550,0
```

```
51120 DRAW 0,0,1:DRAW 0,300,1
51130 FOR i=0 TO 100 STEP 2
51140 MOVE 0,i*3
51150 IF (i/10)=INT(i/10) THEN DRAW -8,i*3,1
51160 DRAW -4,i*3,1
51170 NEXT i
51180 ver=550/anzahl
51190 FOR i=0 TO anzahl
51200 MOVE i*ver,0
51210 DRAW i*550/anzahl,-8,1
51220 NEXT i
51230 LOCATE 1,3
51240 FOR i=100 TO 0 STEP -10
51250 PRINT USING "###";i
51260 PRINT
51270 NEXT i
51280 '
51290 REM ************************************************
51300 REM *                                              *
51310 REM *              Groessten Wert suchen           *
51320 REM *                                              *
51330 REM ************************************************
51340 maximal=wert(1)
51350 FOR i=2 TO anzahl
51360 IF wert(i) > maximal THEN maximal=wert(i)
51370 NEXT i
51380 '
51390 REM ************************************************
51400 REM *                                              *
51410 REM *       Setzen der Werte und ggf. verbinden    *
51420 REM *                                              *
51430 REM ************************************************
51440 faktor=300/maximal
51450 FOR i= 1 TO anzahl
51460 IF wert(i)<0 THEN PRINT CHR$(7):GOTO 51520
51470 MOVE i*ver-2,wert(i)*faktor
51480 DRAW i*ver+2,wert(i)*faktor,1
51490 DRAW i*ver+2,wert(i)*faktor-4,1
51500 DRAW i*ver-2,wert(i)*faktor-4,1
51510 DRAW i*ver-2,wert(i)*faktor,1
51520 NEXT i
51530 IF binden=0 THEN 51590
51540 MOVE 0,0
51550 FOR i=1 TO anzahl
51560 IF wert(i)<0 THEN 51580
51570 DRAW i*ver,wert(i)*faktor,1
51580 NEXT i
51590 LOCATE 7,24
51600 PRINT "Prozentuale Darstellung der Werte  1 bis";
51610 PRINT  anzahl;". Maximalwert :";maximal
51620 RETURN
51630 REM ************************************************
```

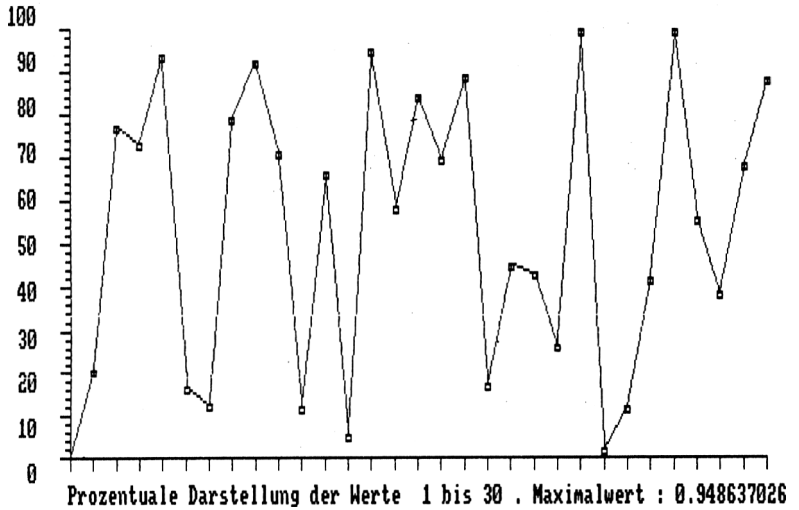
Wenn Sie das Testprogramm vom Balkendiagramm um die Zeile

```
130 binden=1
```

erweitern und die GOSUB-Anweisung in

```
170 GOSUB 51000
```

ändern, können Sie das „Werte-Plot"-Programm damit testen.



7.3 Tortendiagramm

Eine verbreitete Form der grafischen Darstellung ist das sogenannte „Torten- oder Kuchendiagramm". Der Name kommt von der grafischen Aufteilung einer „Torte" (Scheibe) in Stücke, die mit ihrer Größe den jeweilig darzustellenden Wert repräsentieren. Eine Tortengrafik ist nur dann sinnvoll anzuwenden, wenn die Anzahl der darzustellenden Werte maximal 20 ist.

Die gesamte Torte entspricht der Summe aller Tortenstücke und damit auch aller Einzelwerte. Die Größe eines Tortenstückes ist abhängig vom Verhältnis des Einzelwertes zur Summe aller Werte. Deshalb muß als erstes die Summe aller Werte ermittelt werden (52210-52230). Der Ellipsenanteil steht nun im

direkten Verhältnis zur gesamten Ellipse und damit zu 360 Grad. Der Gradanteil eines Tortenstückes errechnet sich dann aus

Gradanteil = wert * (360/summe)

Nacheinander werden alle Winkel in der Hilfsvariablen „help" addiert (52250) und die verschiedenen Summen geben den Winkel des Tortenstückes an, der in Zeile 52270 in die X- und Y-Koordinate umgerechnet und mit einer Linie verbunden wird. Interessant ist noch die Zeile 52300. Ist der Winkel größer als 180 Grad, wird an der Stirnseite der Torte noch ein senkrechter Strich gezogen und die Grafik wirkt somit noch plastischer.

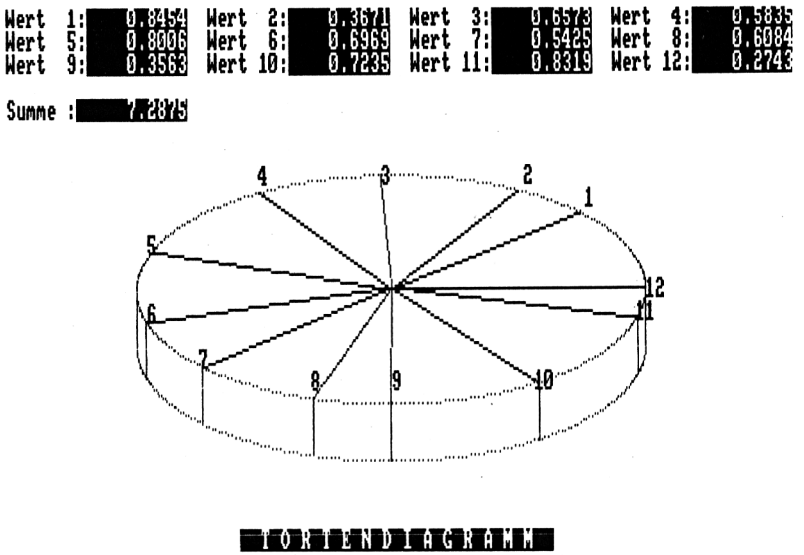
Die Programmzeilen 52080 bis 52120 zeichnen die ungeteilte Torte mit Hilfe des „Ellipsen-Unterprogramms" (siehe 6.3). Dieses Programm müssen Sie zusätzlich mitladen, damit es beim Aufruf (GOSUB 60000) zu keiner Fehlermeldung kommt.

```
52000 REM ************************************ M.W.Thoma **
52010 REM *                  Tortengrafik                 *
52020 REM *                                               *
52030 REM *    wert (...) : Wertarray                     *
52040 REM *    anzahl     : Anzahl der vorhandenen Daten  *
52050 REM *                 (maximal 20)                  *
52060 REM *************************************************
52070 MODE 2:f=1
52080 xm=320:ym=200:xr=200:yr=80:wa=0  :we=360:GOSUB 60000
52090 xm=320:ym=160:xr=200:yr=80:wa=180:we=360:GOSUB 60000
52100 DRAW 520,200,1
52110 MOVE 120,160
52120 DRAW 120,200,1
52130 '
52140 REM *************************************************
52150 REM *                                               *
52160 REM *      Kreisanteil berechnen und zeichnen       *
52170 REM *                                               *
52180 REM *************************************************
52190 summe=0:help=0
52200 PRINT CHR$(22);CHR$(1)
52210 FOR i= 1 TO anzahl
52220 summe=summe+wert(i)
52230 NEXT i
52240 FOR i= 1 TO anzahl
52250 help=help+wert(i)*(360/summe)
52260 MOVE 320,200
52270 DRAW 320+200*COS(help),200+80*SIN(help),1
52280 LOCATE (320+200*COS(help))/8,(200-80*SIN(help))/16
52290 PRINT i
52300 IF help > 180 THEN DRAWR 0,-40,1
52310 NEXT i
```

```
52320 PRINT CHR$(22);CHR$(0)
52330
52340 REM ************************************************
52350 REM *                                              *
52360 REM *        Numerische Ausgabe aller Werte        *
52370 REM *                                              *
52380 REM ************************************************
52390 LOCATE 1,1
52400 FOR i= 1 TO anzahl
52410 PRINT "  Wert";USING " ##";i;
52420 PRINT ":";CHR$(24);USING "#####.####";wert(i);
52430 PRINT CHR$(24);
52440 NEXT i
52450 PRINT
52460 PRINT "  Summe :";CHR$(24);USING "######.####";summe;
52470 LOCATE 26,24
52480 PRINT "  T O R T E N D I A G R A M M  "
52490 PRINT CHR$(24);CHR$(0)
52500 RETURN
52510 REM ************************************************
52520
60000 REM ************************************************
60010 REM *          Ellipse- und Ellipsenteile          *
60020 REM *                                              *
60030 REM *      wa = Winkelanfang ( < we )              *
60040 REM *      we = Winkelende   ( > wa )              *
60050 REM *      xm = X-Koordinate des Mittelpunktes     *
60060 REM *      ym = Y-Koordinate des Mittelpunktes     *
60070 REM *      xr = X-Radius                           *
60080 REM *      yr = Y-Radius                           *
60090 REM *       f = Zeichenfarbe                       *
60100 REM *                                              *
60110 REM ************************************************
60120 DEG
60130 FOR winkel = wa TO we
60140 x=xm+xr*COS(winkel)
60150 y=ym+yr*SIN(winkel)
60160 PLOT x,y,f
60170 NEXT winkel
60180 RETURN
60190 REM ************************************************
```

Testprogramm zum „Tortendiagramm“ mit einem Ausdruck:

```
100 DIM wert(12)
110 anzahl=12
120 FOR z=1 TO anzahl
130 wert(z)=RND(1)*1000
140 NEXT z
150 GOSUB 52000
160 GOTO 160: "Mit ESC-Taste abbrechen
```



7.4 Beschriften von Grafiken - TAG/TAGOFF

Eine Erleichterung zum Beschriften von Grafiken bietet der Befehl

TAG

Nach der TAG-Anweisung (tag = anheften, Kennmarke) werden alle PRINT-Anweisungen an der aktuellen Grafikcursorposition ausgegeben.

```
10 MODE 2
20 TAG
30 MOVE 320,200
40 PRINT "Hallo";
50 END
```

Der Grafikcursor ist mit dem MOVE-Befehl auf X=320 und Y=200 positioniert worden. „Hallo" erscheint auf Grund der TAG-Anweisung (Zeile 20) an dieser Position. Steht hinter der PRINT-Anweisung kein Semikolon (;), so erscheinen zusätzlich noch die normalerweise nicht sichtbaren Steuer- oder Kontrollzeichen (z.B. Return und Line Feed).

Die TAG-Anweisung bleibt solange aktiv, bis eine

TAGOFF

Anweisung folgt. Alle PRINT-Anweisungen beziehen sich dann wieder auf den Textcursor.

```
10 MODE 2
20 FOR x=0 TO 639 STEP 16
30 MOVE x,0: DRAW x,300,1
40 TAG
50 PRINT "*";
60 TAGOFF
70 PRINT "-";
80 NEXT x
90 END
```

Zwei Eigenschaften grenzen die Anwendung der TAG-Anweisung allerdings erheblich ein:

1.) Ein gleichzeitiges aktivieren des Transparent-Modus und des TAG-Modus ist nicht möglich. Gerade beim Beschriften von Grafiken ist dieses aber unerläßlich.

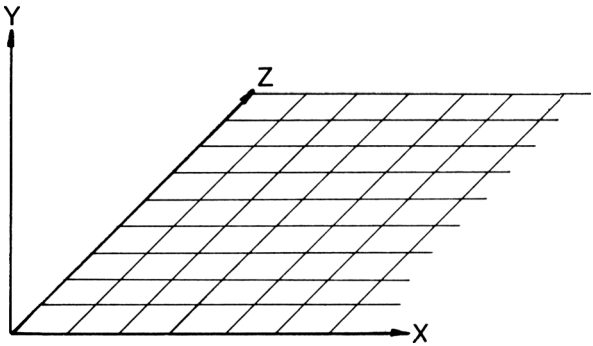
2.) Die Grafikcursorposition wird entsprechend der gePRINTeten Zeichen verschoben, so daß in bestimmten Fällen eine neue Positionierung des Grafikcursors notwendig wird.

Aus diesen Gründen ist auch im Programm zum „Tortendiagramm" (siehe 7.3) die Beschriftung der Torte nicht mit dem TAG-Befehl durchgeführt worden.

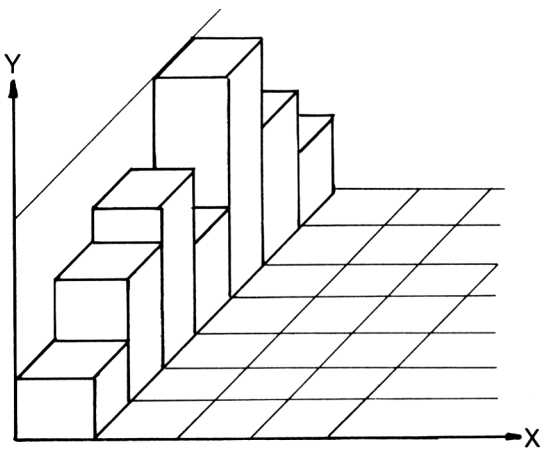
7.5 3D-Balkendiagramm

Bei den bisherigen Programmen zur Darstellung von Werten war es nur möglich, einzelne Werte nebeneinander darzustellen. Damit ein komplexerer Zusammenhang grafisch dargestellt werden kann, ist eine andere Lösung notwendig. Sie wollen z.B. Ihre Haushaltskosten der letzten 8 Monate, aufgegliedert in 9 Warenbereiche, gleichzeitig in einer Grafik überblicken. Neben der X-

und Y-Achse (Anzahl und Größe) wird eine dritte Achse, die räumliche Achse, notwendig.

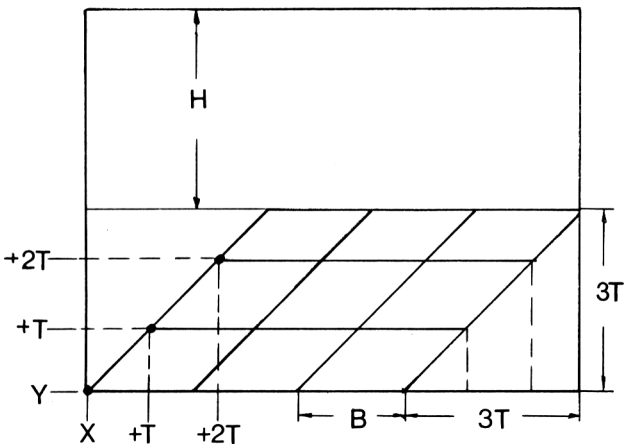


Die erste räumliche Spalte zeigt die 9 Warenbereiche für den ersten Monat an. Die zweite Spalte für den zweiten Monat usw.. Stellen Sie sich nun vor, daß auf jedes so entstandene Feld ein Quader gesetzt wird, der die jeweilige Größe des Wertes anzeigt.



Das Zeichnen eines einzelnen Quaders wurde schon ausführlich beim „Quader"-Programm (siehe 6.8) beschrieben und liegt fertig als Unterprogramm vor. Die Schwierigkeit besteht nun darin, die Quader an die richtige Stelle und

in der richtigen Reihenfolge zu zeichnen. Da ein vornestehender Quader einen Teil des dahinterliegenden Quaders übermalt, muß von hinten nach vorne gezeichnet werden. Weiterhin übermalt ein rechts stehender Quader einen Teil des links daneben liegenden, so daß die Spalten von links nach rechts gezeichnet werden müssen.
Die Berechnung des Standortes eines einzelnen Quaders ist etwas komplizierter, vor allem dann, wenn die Anzahl der darzustellenden Größen variabel gehalten werden soll. Damit alle Werte auf dem Bildschirm Platz finden, muß die Breite und die Tiefe eines Quaders berechnet werden. Weiterhin muß berücksichtigt werden, daß auch die größten Quader an jeder Stelle des Rasters voll auf den Bildschirm passen.



Die Tiefe eines Quaders ist direkt abhängig von der Anzahl der Zeilen. Der zur Verfügung stehende Y-Bereich muß durch die Zeilenanzahl geteilt werden:

Tiefe = Y/Zeilenanzahl

Geht man z.B. davon aus, daß der Y-Bereich 180 Werte groß ist und 3 Zeilenwerte vorhanden sind, beträgt die Tiefe eines Quaders:

T = 180/3
T = 60

Die Breite eines Quaders ist wiederum abhängig von der Anzahl der Spaltenwerte und der Tiefe des Quaders sowie dem zur Verfügung stehenden X-Bereich (639 Punkte):

Breite = (639-Zeilenanzahl*Tiefe)/Spaltenanzahl

Bei 3 Spaltenwerten beträgt die Breite:

B = (639-3*60)/3
B = 153

Das „Quader"-Unterprogramm benötigt als Standort des Quaders die Koordinaten der unteren linken Ecke. Diese Koordinaten sind, wie aus der Zeichnung ersichtlich, innerhalb einer Spalte immer um den Faktor „Tiefe" verschoben. Von Spalte zu Spalte wird dann immer um den Faktor „Breite" erhöht.

Die maximale Höhe der Quader darf nicht den zur Verfügung stehenden Platz überschreiten. Es wird wieder eine Umrechnung zum größten aller Werte durchgeführt (Zeile 53520-53580). Die Übergabevariablen finden Sie in den Remarkzeilen (53000-53120) wieder, wobei noch die Variable „con" erwähnt werden muß. Mit „con" können Sie entscheiden, ob nach jeder gezeichneten Zeile auf eine Tastenbetätigung gewartet werden soll. Hiermit haben Sie die Möglichkeit, sich eine Spalte genauer anzusehen.
Ab Programmzeile 62000 finden Sie das Unterprogramm zum Zeichnen eines Quaders wieder, welches Sie vorher einladen oder später zuladen müssen.

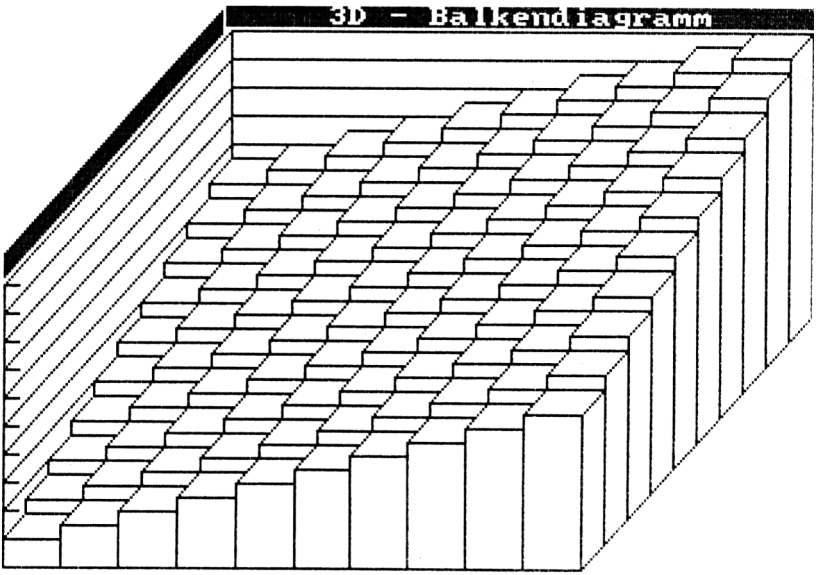
```
53000 REM ************************************ M.W.Thoma **
53010 REM *                                               *
53020 REM *              3D - Balkendiagramm              *
53030 REM *                                               *
53040 REM *  MEX         :  Anzahl der Spaltenwerte        *
53050 REM *  MEY         :  Anzahl der Zeilenwerte         *
53060 REM *  Werte (a,b):  Wertarray                       *
53070 REM *                a = 1 bis MEY                  *
53080 REM *                b = 1 bis MEX                  *
53090 REM *  CON         :  0 = wartet nach jeder Spalte   *
53100 REM *                1 = zeichnet kontinuierlich    *
53110 REM *                                               *
53120 REM *************************************************
53130 MODE 1:INK 0,0:INK 1,10:INK 2,26:INK 3,7:BORDER 0
53140 WINDOW#1,1,6,2,2
53150 '
53160 REM *************************************************
53170 REM *                                               *
53180 REM *          Koordinatenraster zeichnen           *
```

```
53190 REM *                                                  *
53200 REM ************************************************
53210 MOVE 0,0
53220 DRAW 0,200,2
53230 DRAW 180,380,2
53240 DRAW 180,180,2
53250 DRAW 639,180,2
53260 DRAW 639,380,2
53270 DRAW 180,380,2
53280 MOVE 0,0
53290 DRAW 459,0,2
53300 DRAW 639,180,2
53310 MOVE 0,0
53320 DRAW 180,180,2
53330 FOR y=20 TO 180 STEP 20
53340 MOVE 0,y
53350 DRAW 180,180+y,1
53360 DRAW 639,180+y,1
53370 NEXT y
53380 LOCATE 12,1:PEN 3:PRINT CHR$(24);
53390 PRINT "     3D - Balkendiagramm     "
53400 PRINT CHR$(24);:PEN 1
53410 FOR y=206 TO 222 STEP 2
53420 MOVE 0,y
53430 DRAW 172,y+172,3
53440 NEXT y
53450 '
53460 REM ************************************************
53470 REM *                                              *
53480 REM *            Ort der Quader berechnen          *
53490 REM *                                              *
53500 REM ************************************************
53510 tr=1
53520 maximal=wert(1,1)
53530 FOR i=1 TO mey
53540 FOR j=1 TO mey
53550 IF wert(i,j)>maximal THEN maximal=wert(i,j)
53560 NEXT j
53570 NEXT i
53580 ver=200/maximal
53590 t=180/mey
53600 b=(639-mey*t)/mex
53610 FOR sp= 1 TO mex
53620 FOR ze= mey-1 TO 0 STEP -1
53630 x1=((ze+1)*t)+(b*sp)-t-b
53640 y1=ze*t
53650 h=wert(ze+1,sp)*ver
53660 IF wert(ze+1,sp)<0 THEN h=0:PRINT CHR$(7);:GOTO 53670
53670 GOSUB 62000
53680 NEXT ze
53690 in$=INKEY$:IF con =1 THEN 53740
53700 IF in$ <> "" THEN PAPER#1,0:CLS#1:GOTO 53740
53710 f=F+2:FOR i=1 TO 400:NEXT
53720 IF f>2 THEN f=0
53730 PAPER#1,f:CLS#1:GOTO 53690
```

```
53740 NEXT sp
53750 '
53760 REM ************************************************
53770 REM *                                              *
53780 REM *                Skala zeichnen                *
53790 REM *                                              *
53800 REM ************************************************
53810 FOR y= 20 TO 200 STEP 20
53820 MOVE 0,y
53830 DRAW 12,y,2
53840 NEXT y
53850 RETURN
53860 REM ************************************************
53870 '
62000 REM ************************************************
62010 REM *                                              *
62020 REM *    Zeichnen von (ausgefuellten) Quadern      *
62030 REM *                                              *
62040 REM * X1   : X-Koordinate Quader unten              *
62050 REM * Y1   : Y-Koordinate Quader links              *
62060 REM * H    : Hoehe des Quaders (0-200 Bildpunkte)   *
62070 REM * B    : Breite des Quaders (Bildpunkte)        *
62080 REM * T    : Tiefe des Quaders (Bildpunkte)         *
62090 REM * TR   : 0 = Quader transparent darstellen      *
62100 REM *        1 = Quader farbig ausfuellen           *
62110 REM *                                              *
62120 REM ************************************************
62130 MOVE X1,Y1
62140 DRAW X1,Y1+H,2
62150 DRAW X1+B,Y1+H,2
62160 DRAW X1+B,Y1,2
62170 DRAW X1,Y1,2
62180 MOVE X1,Y1+H
62190 DRAW X1+T,Y1+H+T,2
62200 DRAW X1+B+T,Y1+H+T,2
62210 DRAW X1+B,Y1+H,2
62220 MOVE X1+B,Y1
62230 DRAW X1+B+T,Y1+T,2
62240 DRAW X1+B+T,Y1+H+T,2
62250 IF tr=0 THEN 62360
62260 FOR S=Y1+2 TO Y1+H-2 STEP 2
62270 MOVE X1+2,S
62280 DRAW X1+B-2,S,1
62290 MOVER 4,2
62300 DRAW X1+B-2+T,S+T-2,1
62310 NEXT S
62320 FOR S=X1+4 TO X1+B STEP 2
62330 MOVE S,Y1+H+2
62340 DRAW S+T-4,Y1+H+T-2,3
62350 NEXT S
62360 RETURN
62370 REM ************************************************
```

Testprogramm zum „3D-Balkendiagramm“:

```
100 DIM wert(10,10)
110 mex=10:mey=10:con=1
120 FOR i=1 TO mey
130 FOR j=1 TO mex
140 wert(i,j)=i+j
150 NEXT j
160 NEXT i
170 GOSUB 53000
180 GOTO 180
```



Mit dem 3D-Balkendiagramm soll die Unterprogrammbibliothek abgeschlossen werden. Insgesamt sind eine Reihe von Programmen entstanden, die bei vielen Anwendungen nützliche Dienste leisten können. Benötigen Sie andere Grafikroutinen, können Sie sich diese jetzt ohne weiteres selber entwickeln.

8.0 Das Grafikfenster

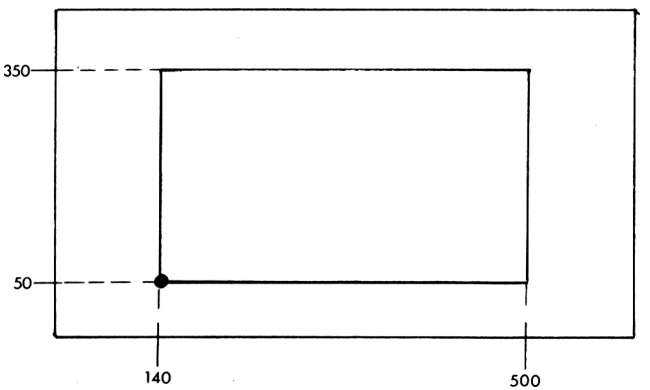
Im Normalzustand (Reset) können auf dem gesamten sichtbaren Bildschirm einzelne Punkte (Linien) in verschiedenen Farben gesetzt werden. Liegt der zu zeichnende Punkt außerhalb des Bildschirms (z.B. X=700;Y=500), wird er automatisch ignoriert (keine Fehlermeldung). Der Bildschirmbereich, in dem Punkte gesetzt werden können, kann mit Hilfe des bekannten ORIGIN-Befehls verändert werden. Nach den Angaben für den Ursprung (Nullpunkt) folgen 4 weitere Parameter, die die Größe des Bereiches festlegen, in dem Punkte sichtbar sind.

ORIGIN xn, yn, xa, xe, ya, ye

„xn" und „yn" bestimmen - wie gewohnt - den Nullpunkt des Koordinatensystems. „xa"/„xe" geben den X-Koordinatenbereich und „ya"/„ye" den Y-Bereich an, in dem ein Punkt sichtbar ist. Diese Koordinaten beziehen sich immer auf das normale (640;400) Koordinatensystem des Bildschirms. Verkleinern Sie den Bereich, in dem Grafik sichtbar sein kann, mit

ORIGIN 140,100,140,500,100,350

so erhalten Sie folgendes Grafikfenster:



Die außerhalb des Grafikfensters liegenden Bildschirmpunkte können dann nicht mehr gesetzt werden. Alle Grafik-Funktionen beziehen sich ab sofort nur

noch auf das definierte Fenster. Auch das Grafikfenster bleibt bis zu einem Reset oder einem MODE-Befehl in der definierten Größe. Ein normaler ORIGIN-Befehl

ORIGIN 320,200

hat keinerlei weiteren Einfluß auf die Größe des Grafikfensters, sondern bestimmt nur den Nullpunkt im Grafikfenster neu. Das Fenster kann „aufgehoben" werden, indem es seine ursprüngliche Größe von

ORIGIN 0,0,0,639,0,399

erhält.

Ein Grafikfenster ist immer dann sinnvoll, wenn auf dem gesamten Bildschirm nur ein Teil für Grafikausgaben bestimmt sein soll (Werte und grafische Darstellung der Werte).
Unabhängig vom Bildschirm läßt sich das Grafikfenster mit dem Befehl

CLG faßnummer

mit einer Farbe füllen (löschen). Damit wird im Gegensatz zum „CLS" nur der Bereich gelöscht, der für die Grafik zuständig ist. Im Einschaltzustand (Reset und MODE), füllt ein „CLG"-Befehl den gesamten Bildschirm mit einer Farbe (auch alle Textfenster) aus, da das Grafikfenster den gesamten Bildschirm umfaßt. Je nach gewähltem Bildschirmmodus kann das Grafikfenster mit 2, 4 oder 16 Farben gefüllt werden.

Neben der Begrenzung des Grafikzeichenbereiches bietet sich der modifizierte ORIGIN-Befehl in Verbindung mit CLG zum schnellen Einfärben größerer Bereiche an. Es lassen sich nicht nur dicke Bildschirmrahmen oder Flaggen (Fahnen) leicht realisieren, sondern auch einfache Balkendiagramme etc. Das Beispielprogramm erzeugt farbige Streifen in vertikaler und horizontaler Richtung, wobei die Farbe zufällig bestimmt wird.

```
1000 REM ************************************ M.W.Thoma **
1010 REM *                                               *
1020 REM *          Grafik mit dem Grafikfenster         *
1030 REM *                                               *
1040 REM *************************************************
1050 MODE 0
1060 FOR x=0 TO 630 STEP 10
1070 ORIGIN 0,0,x,x+10,0,399
```

```
1080 CLG INT(RND(1)*13)+1
1090 NEXT x
1100 FOR t=1 TO 1000: NEXT t
1110 FOR y=0 TO 399 STEP 10
1120 ORIGIN 0,0,0,639,y,y+10
1130 CLG INT(RND(1)*13)+1
1140 NEXT y
1150 FOR t=1 TO 1000: NEXT t
1160 GOTO 1060
1170 REM ***************************************************
```

9.0 Funktions-Plot

Geradezu klassisch ist die Anwendung von Computergrafik bei der Darstellung von mathematischen Funktionen jeder Art und darum darf sie auch hier nicht fehlen. Sicherlich ist Ihnen aus der Schule noch das Erstellen von Funktionen auf Millimeterpapier (unliebsam) bekannt. Alle Koordinaten mußten nach der jeweiligen Funktion mühsam ausgerechnet und in die Zeichnung übertragen werden. Diese Arbeit kann der CPC-464 ohne weiteres für Sie übernehmen.

Das Programm „Funktions-Plot" erlaubt es, jede Funktion in ein wahlfreies Koordinatensystem zu zeichnen. Weiterhin steht es Ihnen frei, das Koordinatensystem und eine Markierung auf der X- und Y-Achse zu zeichnen.

Der Grafikzeichenbereich wird in der Zeile 1180 auf ein 400*200 Feld begrenzt, so daß in X-Richtung 400 und in Y-Richtung 100 (200 Werte) Punkte zur Verfügung stehen. Jedes gewählte Koordinatensystem muß in dieses Fenster passen und entsprechend umgerechnet werden. Die Anpassung des gewählten Koordinatensystems an das Grafikfenster findet in den Programmzeilen 1930 bis 1950 statt. Zunächst werden zwei Verhältnisvariablen (vx, vy) für die X- und Y-Achse berechnet. „xu" und „xo" bestimmen die untere und obere Grenze des zu zeichnenden Koordinatensystems. Dieser Bereich muß vollständig auf dem Bildschirm (400 X-Werte) sichtbar sein. Die Länge der X-Achse berechnet sich dann aus:

X-Länge = xo - xu
X-Länge = +5 - (-5)
X-Länge = 10

Wird ein „xo" von +5 und ein „xu" von -5 angenommen, so beträgt die Länge der X-Achse 10 Werte. Dieser Wert wird dann dem Bildschirmsystem angepaßt:

vx = 400/(xo-xu)
vx = 400/10
vx = 40

Jeder tatsächliche X-Wert (+5 bis -5) muß mit dem Faktor „vx" multipliziert werden, damit er an die richtige Stelle des Bildschirms gesetzt wird. Der Nullpunkt der X-Achse befindet sich an der Stelle

xn = 520 - xo*vx
xn = 520 - +5*40
xn = 520 - 200
xn = 320

und damit genau in der Mitte des Bildschirms (-5...0...+5). Die „520" ist dabei die letzte auf dem Bildschirm nutzbare X-Koordinate (siehe Programmzeile 1180). Der kleinste darstellbare X-Wert (-5) liegt damit bei

x = -5 * vx
x = -5 * 40
x = -200

und somit noch im Grafikfenster. Der größte Wert (+5) liegt ebenfalls noch im Grafikfenster (5*40=200). Die Y-Anpassung ist vollständig identisch und findet in der Programmzeile 1940 statt. Anschließend wird der Nullpunkt beider Achsen (Zeile 1950) bestimmt.

Hiermit sind alle notwendigen Vorbereitungen getroffen, um jede Funktion in dem Grafikfenster darzustellen. Das eigentliche Zeichnen der Funktion findet in den Programmzeilen 2000 bis 2020 statt. Die Y-Werte werden nacheinander mit der Funktion (fn f(x)) im Bereich „xu" bis „xo" berechnet und mit den Faktoren „vx" und „vy" dem Bildschirm angepaßt. Die Schrittweite der Berechnung ist mit

(xo-xu)/400

so gewählt, daß jeder X-Bildschirmpunkt gesetzt wird.

Ein besonderes Problem (nicht nur beim CPC-464) stellt die Übergabe von Funktionen in ein Programm dar. In der Programmzeile 1260 befindet sich die darzustellende Funktion (sin(x)). Damit eine andere Funktion berechnet werden kann, muß diese Programmzeile geändert werden. Die einfachste aber auch unkomfortabelste Lösung besteht darin, daß vor jedem Start des Programms die Funktion geändert wird (Zeile neu schreiben). Dadurch müßte das Programm immer wieder unterbrochen werden. Wesentlich komfortabler wäre es, die Funktion direkt über eine Eingabe im Programm zu übergeben. Dieses ist allerdings nur mit einem Trick möglich: Das Programm muß so unterbrochen werden, daß der Bediener es nicht „merkt". Doch sehen Sie sich dazu die Programmzeilen 1480 bis 1580 an. Die neue Funktion wird zunächst in die Stringvariable „funk$" mit Hilfe einer INPUT-Anweisung übergeben.

Anschließend wird in der Stringvariable „x$" eine komplette Programmzeile abgespeichert, die dann auf die Taste 139 (kleine Entertaste) gelegt wird (das Belegen der Tasten finden Sie im Kapitel 15.3). Wurde z.B. in die Variable „funk$" die Funktion „cos(x)" eingegeben, so liegt auf der kleinen Entertaste die Zeile:

```
1260 DEF FN f(x)=COS(x)  (Return) GOTO 1260  (Return)
```

Nach der Aufforderung, die kleine Entertaste zu betätigen (Zeile 1520 - 1560), wird das Programm mit STOP unterbrochen. Betätigen Sie nun die kleine Entertaste, wird die obige Zeile auf dem Bildschirm ausgegeben und mit dem ersten Return (CHR$(13)) ins Programm übergeben. Das folgende „GOTO 1260" wird ebenfalls ausgePRINTet und mit dem Return direkt ausgeführt. Das Programm wird ab der Zeile 1260 fortgesetzt. Damit das Ausdrucken der neuen Programmzeile und die GOTO-Anweisung nicht sichtbar ist, wird die Zeichenfarbe vorher (Zeile 1570) gleich der Hintergrundfarbe gesetzt. Das Fortsetzen des Programms ab der Zeilennummer 1260 ist umbedingt notwendig, damit die neue Funktion auch tatsächlich bestimmt wird.

Weiterhin ist in dem Programm zum Funktions-Plot eine umfassende Fehlerbehandlung am Ende des Programms eingebaut. Diese Fehlerbehandlung ist zum einen notwendig, um Fehler bei der Funktionseingabe zu erkennen und zum anderen, damit es bei mathematisch nicht lösbaren Aufgaben zu keiner Fehlermeldung und keinem Programmabbruch kommt (Teilung durch Null, negative Wurzel, overflow). Tritt einer dieser Fehler auf, wird eine Meldung ausgegeben, das Zeichnen allerdings fortgesetzt. In diesem Fall können Sie das Zeichnen mit dem Betätigen einer Taste unterbrechen. Tritt ein anderer Fehler auf (z.B. SYNTAX ERROR), kann der Fehler bei einem einwandfreien Programm nur in der Funktionseingabe liegen. Eine entsprechende Meldung weist darauf hin, daß Sie die Funktion überprüfen und neu eingeben können (Die Fehlerbehandlung ist im Kapitel 17.0 ausführlich beschrieben).

```
1000 REM ************************************ M.W.Thoma **
1010 REM *                                               *
1020 REM *                Funktions-Plot                 *
1030 REM *                                               *
1040 REM *************************************************
1050 MODE 2
1060 INK 0,0:INK 1,26:BORDER 0
1070 funk$="sin(x)"
1080 xu=-10:xo=10:yu=-1.5:yo=1.5
1090 LOCATE 30,17
1100 PRINT "F u n k t i o n s p l o t"
```

```
1110 WINDOW#0,1,80,20,25
1120 WINDOW#1,1,80,1,2
1130 WINDOW#2,68,80,4,16
1140 ORIGIN 320,250
1150 MOVE 208,106
1160 DRAW -208,106,1:DRAW -208,-106,1
1170 DRAW 208,-106,1:DRAW 208,106,1
1180 ORIGIN 120,150,120,520,350,150
1190 PAPER#2,1:PEN#2,0:CLS#2
1200 '
1210 REM ************************************************
1220 REM *                                              *
1230 REM *                    M e n u                   *
1240 REM *                                              *
1250 REM ************************************************
1260 DEF FN f(x)=SIN(x/2)+COS(x)
1270 PEN 1
1280 CLS:CLS#1
1290 ON ERROR GOTO 2200
1300 KEY 139,CHR$(13)
1310 tt=FN f(1)
1320 LOCATE#1,1,2
1330 PRINT#1," > Funktion :  f(x)=";funk$
1340 CLS
1350 INPUT " <A>lte oder <N>eue Funktion oder <E>nde ";in$
1360 IF UPPER$(in$)="A" THEN 1600
1370 IF UPPER$(in$)="N" THEN 1420
1380 IF UPPER$(in$)="E" THEN MODE 2: END
1390 SOUND 1,500,10
1400 GOTO 1340
1410 '
1420 REM ************************************************
1430 REM *                                              *
1440 REM *               Neue Funktion eingeben         *
1450 REM *                                              *
1460 REM ************************************************
1470 PRINT
1480 INPUT "  Neue Funktion eingeben :";funk$
1490 x$="1260 def fn f(x)="+funk$+CHR$(13)
1500 x$=x$+"goto 1260"+CHR$(13)
1510 KEY 139,x$
1520 CLS
1530 PRINT CHR$(24)
1540 PRINT "        Bitte betaetigen Sie die        "
1550 PRINT "        die kleine <ENTER>-Taste        "
1560 PRINT CHR$(24);
1570 PEN 0
1580 STOP
1590 '
1600 REM ************************************************
1610 REM *                                              *
1620 REM *          Koordinatensystem bestimmen         *
1630 REM *                                              *
1640 REM ************************************************
1650 CLS
```

```
1660 INPUT " Koordinaten neu bestimmen (j/n) ";in$
1670 IF UPPER$(in$) <>"J" THEN 1740
1680 INPUT " X-Koordinate links  :";xu
1690 INPUT " X-Koordinate rechts :";xo
1700 INPUT " Y-Koordinate unten  :";yu
1710 INPUT " Y-Koordinate oben   :";yo
1720 IF xu>=xo THEN PRINT CHR$(7):CLS:GOTO 1680
1730 IF yu>=yo THEN PRINT CHR$(7):CLS:GOTO 1680
1740 CLS#2
1750 PRINT#2, " Koordinaten:"
1760 PRINT#2, " XU =";xu
1770 PRINT#2, " XO =";xo
1780 PRINT#2, " YU =";yu
1790 PRINT#2, " YO =";yo
1800 CLS
1810 INPUT " > Koordinatensystem zeichnen (j/n) ";ko$
1820 INPUT " > Markierungen setzen        (j/n) ";ma$
1830 IF UPPER$(ma$) <>"J" THEN 1860
1840 INPUT " > X-Marke :";xm
1850 INPUT " > Y-Marke :";ym
1860 INPUT " > Bildschirm loeschen        (j/n) ";lo$
1870 '
1880 REM ************************************************
1890 REM *                                              *
1900 REM * Zeichnen der Koordinaten, Markierungen, Kurve *
1910 REM *                                              *
1920 REM ************************************************
1930 vx=400/(xo-xu)
1940 vy=200/(yo-yu)
1950 ORIGIN 520-xo*vx,350-yo*vy
1960 IF UPPER$(lo$)="J" THEN CLG
1970 IF UPPER$(ko$)="J" THEN GOSUB 2040
1980 IF UPPER$(ma$)="J" THEN GOSUB 2100
1990 PRINT#2
2000 FOR x=xu TO xo STEP (xo-xu)/400
2010 PLOT x*vx,FN f(x)*vy,1
2020 NEXT x
2030 GOTO 1280
2040 '
2050 MOVE xu*vx,0
2060 DRAW xo*vx,0,1
2070 MOVE 0,yo*vy
2080 DRAW 0,yu*vy,1
2090 RETURN
2100 '
2110 MOVE xm*vx,0
2120 DRAW xm*vx,-6,1
2130 MOVE -6,ym*vy
2140 DRAW 8,ym*vy,1
2150 PRINT#2
2160 PRINT#2, " XM =";xm
2170 PRINT#2, " YM =";ym
2180 RETURN
2190 '
2200 REM ************************************************
```

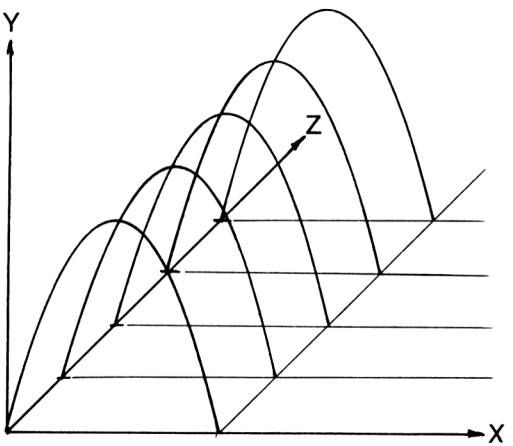
```
2210 REM *                                                  *
2220 REM *                  Fehlerbehandlung                *
2230 REM *                                                  *
2240 REM ***************************************************
2250 PRINT CHR$(7);
2260 CLS
2270 LOCATE 5,3
2280 in$=INKEY$
2290 IF in$<>"" THEN 2400
2300 IF ERR=5 THEN PRINT " Improper argument !":RESUME NEXT
2310 IF ERR=6 THEN PRINT " Overflow          !":RESUME NEXT
2320 IF ERR=11 THEN PRINT " Division by zero !":RESUME NEXT
2330 PRINT CHR$(24);
2340 LOCATE 1,3
2350 PRINT "          Voraussichtlich Fehler            "
2360 PRINT "         bei der Funktionseingabe !         "
2370 PRINT CHR$(24);
2380 SOUND 1,800,200,5
2390 FOR i=1 TO 5000: NEXT i
2400 RESUME 1320
2410 REM ***************************************************
```

Hier ein paar Funktionen zum Testen des Funktions-Plot:

	Funktion	xu	xo	yu	yo
1	cos(x)	-10	10	-1.5	-1.5
2	sin(x/2)+cos(x)	-20	20	-3	3
3	cos(x)+cos(2*x)+cos(5*x)	-2	2	-3	3
		-5	5	-3	3

10.0 3D-Funktions-Plot

Wie schon beim „Quader-Programm" vorbereitend erläutert wurde, ist die räumliche Darstellung eines Körpers oder aber einer 3-dimensionalen Funktion wesentlich komplizierter als die Darstellung eines Körpers (2-dimensional). 3D-Funktionen sind nun diese „Gebirgslandschaften", die Ihnen sicherlich aus der Werbung für alle möglichen Computer bekannt sind. Anscheinend ist es heute immer noch so, daß solch spektaluläre Grafiken ein Augenfang für viele Programmierer (Käufer) sind.
3-dimensionale Funktionen sind nicht nur von den bekannten X- und Y-Achsen abhängig, sondern auch noch von einer räumlichen Z-Achse. Das Prinzip der 3-dimensionalen Darstellung beruht auf dem Hintereinanderlegen mehrerer 2-dimensionaler Kurven.



Bei der ersten Kurve ist „z" konstant auf „0", dann „z" auf „1" usw. und es wird eine normale Funktion gezeichnet. Damit ein räumlicher Effekt zustande kommt, werden die einzelnen Kurven um 45 Grad versetzt gezeichnet, was zur Folge hat, daß die Kurven nach rechts oben verschoben werden. Was also berechnet werden muß, ist der Versatz nach rechts oben in Abhängigkeit von der Z-Achse (Schrittweite der Z-Achse), d.h. der Versatz in X- und Y-Richtung.

Der Versatz in X- und Y-Richtung errechnet sich aus:

$$cx = z * \cos(45)$$
$$cy = z * \sin(45)$$

Da sich bei einer räumlichen Darstellung von 45 Grad die Tiefe halbiert, muß „z" noch halbiert werden, um den richtigen Wert zu erhalten:

$$cx = z/2 * \cos(45)$$
$$cy = z/2 * \sin(45)$$

Da die Y-Koordinaten schon doppelt belegt sind (200 Punkte = 400 Werte), braucht die Y-Koordinate nicht mehr halbiert zu werden.

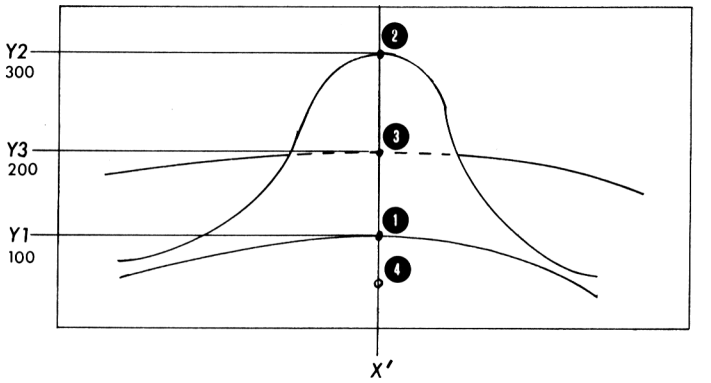
$$cy = z / \sin(45)$$

Dieser X- und Y-Versatz ist den berechneten Koordinaten hinzuzuaddieren. Die so erhaltenen X- und Y-Werte müssen dann (wie in 7.1 beschrieben) wieder dem Bildschirmsystem angepaßt werden.

10.1 Sichtbar oder unsichtbar?

Ein entscheidendes Problem bei der 3-dimensionalen Funktionsdarstellung ist, daß Teile des Funktionsplots nicht sichtbar sind, weil sie von einem „vorderen" Teil verdeckt sind. Damit eine einwandfreie Betrachtung der Funktion möglich ist, dürfen verdeckte Funktionsteile nicht sichtbar sein. Zur Lösung dieses Problems gibt es viele Möglichkeiten, wobei mir die Anregung eines Freundes am Besten gefällt. Sie geht davon aus, daß der Computer sich merkt, ob in einem bestimmten Bereich schon ein Punkt gesetzt worden ist. Sehen sie sich dazu einmal folgende Zeichnung an:

Für jede Spalte (X-Koordinate) wird der größte und der kleinste bisher gezeichnete Punkt (Y-Koordinate) vermerkt (P1;P2). Liegt nun ein zu zeichnender Punkt (P3) zwischen dem größten und kleinsten Wert, so liegt er „hinter" der gezeichneten Funktion und darf daher nicht gezeichnet werden. Liegt er oberhalb P2 oder unterhalb P1, so ist er sichtbar und muß gezeichnet werden (P4). Gleichzeitig stellt dann der Punkt P4 den neuen kleinsten Wert dar.



Für jede X-Koordinate (Spalte) müssen zwei Kontrollvariablen vorhanden sein, die den jeweiligen kleinsten und größten Wert vermerken. Dafür bietet sich ein 2-dimensionales Array an:

crtl (0,x) kleinster Wert bei x
crtl (1,x) größter Wert bei x

Bevor ein Punkt gesetzt und damit sichtbar wird, muß überprüft werden, ob dieser Punkt größer bzw. kleiner als der jeweilige Kontrollwert ist. Nur wenn dieses zutrifft, darf er gezeichnet werden. Ist dieser Punkt z.B. kleiner als der Kontrollwert „crtl (0,x)", entspricht dieser Wert dem neuen kleinsten Wert. Dieser Wert muß dann in die Variable crtl (0,x) gelegt werden.
Damit es beim Zeichnen der ersten Ebene (erster Z-Achsen-Wert) zu keinem Problem kommt (darf dieser Punkt gezeichnet werden??), bediene ich mich hier eines einfachen Tricks (Programmzeilen 2290-2310). Bevor das Zeichnen beginnt, werden alle Kontrollvariablen auf einen endlich großen (1E+38) oder einen endlich kleinen (-1E+38) Wert gelegt. (Die Kontrollvariablen für die kleinsten Werte (crtl (0,x)) auf endlich groß und für die größten Werte (crtl (1,x)) auf endlich klein!) Damit ist gewährleistet, daß die erste Funktionsebene auf jeden Fall gezeichnet wird.

Das Programm „3D-Funktionsplot" ist sehr variabel gehalten, so daß einige Eingaben notwendig sind. Sie können das Koordinatensystem frei wählen, wobei

xu;xo = X-Koordinatenbereich (z.B. xu= -5;xo= +5)
yu;yo = Y-Koordinatenbereich (z.B. yu= -2;yo= +5)
zu;zo = Z-Koordinatenbereich (z.B. zu= -1;zo= +1)

entsprechen und das gewählte Koordinatensystem den gesamten sichtbaren Bildschirmbereich ausfüllt. Innerhalb dieses Koordinatensystem können Sie einen Bereich auswählen, in dem die Funktion gezeichnet werden soll.

xa;xe = Y-Zeichenbereich von/bis (z.B. xa= 0;xe= +4)
za;ze = Z-Zeichenbereich von/bis (z.B. za= 0;ze= +1)

Dieser Zeichenbereich muß selbstverständlich innerhalb des gewählten Koordinatensystems liegen.
Weiterhin ist es möglich, die Schrittweite der zu berechnenden Funktion variabel zu halten. Bei einer X-Schrittweite von '1' wird jeder mögliche X-Punkt berechnet und gezeichnet, bei einer X-Schrittweite von '8' entsprechend nur jeder 8. Punkt. Die Zeichnungen werden zwar bei einer größeren Schrittweite nicht ganz so „fein", dies hat allerdings den Vorteil, daß die Zeichnung wesentlich schneller fertiggestellt wird, da nur jeder 8. Punkt berechnet und gegebenenfalls gesetzt werden muß.

Abschließend ist noch zu bestimmen, ob ein Koordinatensystem zu zeichnen ist. Mit Hilfe des Koordinatensystems ist eine bessere Zuordnung der Wertgrößen möglich. Sie können außerdem noch entscheiden, ob nur die sichtbaren Punkte gesetzt werden sollen oder ob eine normale Darstellung ausreichend ist. Bei der „normalen" Darstellung werden alle Funktionen übereinander gezeichnet. Hierdurch entsteht ein erheblicher Geschwindigkeitsvorteil, da keine Überprüfung der Kontrollvariablen notwendig ist.

Die Eingabe neuer Funktionen sowie die Fehlerbehandlung ist identisch wie beim „Funktions-Plot" in Kapitel 9.0.

```
1000 REM ************************************ M.W.Thoma **
1010 REM *                                               *
1020 REM *              3D - Funktions-Plot              *
1030 REM *                                               *
1040 REM *****************************************************
1050 MODE 2
1060 INK 0,0:INK 1,26:BORDER 0
1070 DIM crtl(1,700)
1080 funk$="sin(x*z)"
1090 WINDOW#0,1,39,20,25
1100 WINDOW#1,41,80,20,25
1110 WINDOW#2,1,80,1,2
```

```
1120 PAPER 1:PEN 0:CLS
1130 PAPER#1,1:PEN#1,0:CLS#1
1140 ORIGIN 14,106
1150 DRAW 0,260,1:DRAW 610,260,1
1160 DRAW 610,0,1:DRAW 0,0,1
1170 ORIGIN 20,112,20,620,362,112
1180 xu=-1:xo=6:yu=-1:yo=6:zu=-1:zo=6
1190 '
1200 REM ***************************************************
1210 REM *                                                 *
1220 REM *                    M e n u                      *
1230 REM *                                                 *
1240 REM ***************************************************
1250 ON ERROR GOTO 2600
1260 CLS#2
1270 PRINT#2,TAB(32);"3D - FUNKTIONSPLOT"
1280 KEY 139,CHR$(13)
1290 PEN 0
1300 LOCATE#2,3,2
1310 PRINT#2,"Funktion : f(x)=";funk$
1320 DEF FN f(x)=SIN(x*z)
1330 TT=FN f(1)
1340 '
1350 CLS
1360 PRINT"(A)lte oder (N)eue Funktion oder (E)nde";
1370 LOCATE 2,2
1380 INPUT "> ";in$
1390 IF UPPER$(in$)="E" THEN MODE 2:PRINT CHR$(24):END
1400 IF UPPER$(in$)="A" THEN 1670
1410 IF UPPER$(in$)="N" THEN 1450
1420 SOUND 1,800,5
1430 GOTO 1370
1440 '
1450 REM ***************************************************
1460 REM *                                                 *
1470 REM *              Neue Funktion eingeben             *
1480 REM *                                                 *
1490 REM ***************************************************
1500 PRINT " > Neue Funktion eingeben :"
1510 INPUT " >";funk$
1520 x$="1320 def fn f(x)="+funk$+CHR$(13)
1530 x$=x$+"goto 1250"+CHR$(13)
1540 KEY 139,x$
1550 CLS
1560 LOCATE 1,2
1570 PRINT "        Bitte betaetigen Sie die"
1580 PRINT "        die kleine <ENTER>-Taste"
1590 PEN 1
1600 STOP
1610 '
1620 REM ***************************************************
1630 REM *                                                 *
1640 REM *          Koordinatensystem bestimmen            *
1650 REM *                                                 *
1660 REM ***************************************************
```

```
1670 LOCATE 2,2
1680 INPUT "> Koordinaten neu bestimmen ";in$
1690 IF UPPER$(in$) <>"J" THEN 1780
1700 INPUT " > XU, XO :";xu,xo
1710 INPUT " > YU, YO :";yu,yo
1720 INPUT " > ZU, ZO :";zu,zo
1730 IF xu<xo AND yu<yo AND zu<=zo THEN 1780
1740 SOUND 1,800.2
1750 CLS
1760 GOTO 1700
1770 '
1780 CLS:CLS#1
1790 PRINT#1," >XU/XO= ";USING "########.######";xu,xo
1800 PRINT#1," >YU/YO= ";USING "########.######";yu,yo
1810 PRINT#1," >ZU/ZO= ";USING "########.######";zu,zo
1820 '
1830 CLS
1840 INPUT " > Koordinatensystem zeichnen ";ko$
1850 CLS
1860 INPUT " > Z-Schrittweite  :";zw
1870 INPUT " > X-Schrittweite  :";xw
1880 CLS
1890 PRINT"      Zeichenbereich innerhalb des"
1900 PRINT"      Koordinatensystems bestimmen"
1910 INPUT " > XA, XE  :";xa,xe
1920 INPUT " > ZA, ZE  :";za,ze
1930 IF xa>=xe OR za>=ze THEN 1880
1940 INPUT " Nur sichtbare Punkte zeichnen ";si$
1950 '
1960 PRINT#1," >XA/XE= ";USING "########.######";xa,xe
1970 PRINT#1," >ZA/ZE= ";USING "########.######";za,ze
1980 PRINT#1," >ZW/XW= ";USING "########.######";zw,xw
1990 '
2000 REM ************************************************
2010 REM *                                              *
2020 REM *   Zeichnen der Koordinaten und der Funktion  *
2030 REM *                                              *
2040 REM ************************************************
2050 vx=600/(xo-xu)
2060 vy=250/(yo-yu)
2070 cx=COS(45)*2
2080 cy=SIN(45)
2090 z1=zo/cx:z2=zu/cx
2100 sz=(z1-z2)/(z1*vx-z2*vx)
2110 nx=640-xo*vx
2120 ORIGIN 620-xo*vx,362-yo*vy
2130 CLG
2140 '
2150 IF UPPER$(ko$) <>"J" THEN 2230
2160 MOVE xu*vx,0
2170 DRAW xo*vx,0,1
2180 MOVE 0,yo*vy
2190 DRAW 0,yu*vy,1
2200 MOVE zu/cx*vx,zu/cy*vy
2210 DRAW zo/cx*vx,zo/cy*vy,1
```

```
2220 '
2230 CLS
2240 LOCATE 10,2
2250 PRINT "Zeichne Funktion."
2260 PRINT "    Ich bitte um etwas Geduld !"
2270 IF UPPER$(si$) <>"J" THEN ns=1:GOTO 2330
2280 '
2290 FOR i=0 TO 640
2300 crtl(0,i)=1E+38:crtl(1,i)=-1E+38
2310 NEXT i
2320 '
2330 FOR z=za TO ze STEP zw*sz*4
2340 FOR x=xa TO xe STEP (xo-xu)/(600/xw)
2350 px=(x+z/cx)*vx:py=(FN f(x)+z/cy)*vy
2360 IF px>xo*vx OR px<xu*vx THEN 2410
2370 IF ns=1 THEN 2400
2380 IF py<crtl(0,px+nx) OR py>crtl(1,px+nx) THEN 2510
2390 GOTO 2410
2400 PLOT px,py,1
2410 NEXT x
2420 NEXT z
2430 ns=0
2440 GOTO 1350
2450 '
2460 REM ***************************************************
2470 REM *                                                 *
2480 REM *            Pruefen ob sichbarer Punkt           *
2490 REM *                                                 *
2500 REM ***************************************************
2510 IF py<crtl(0,px+nx) THEN crtl(0,px+nx)=py:GOTO 2530
2520 IF py>crtl(1,px+nx) THEN crtl(1,px+nx)=py
2530 IF xw=1 THEN 2400
2540 FOR i=px+nx TO px+nx+xw
2550 IF py<crtl(0,i) THEN crtl(0,i)=py
2560 IF py>crtl(1,i) THEN crtl(1,i)=py
2570 NEXT i
2580 GOTO 2400
2590 '
2600 REM ***************************************************
2610 REM *                                                 *
2620 REM *                  Fehlerbehandlung               *
2630 REM *                                                 *
2640 REM ***************************************************
2650 PRINT CHR$(7);
2660 LOCATE#2,3,1
2670 in$=INKEY$
2680 IF in$<>"" THEN 2780
2690 IF ERR=5 THEN PRINT#2, "Improper argument!":RESUME NEXT
2700 IF ERR=6 THEN PRINT#2, "Overflow         !":RESUME NEXT
2710 IF ERR=11 THEN PRINT#2,"Division by zero !":RESUME NEXT
2720 CLS
2730 LOCATE 1,3
2740 PRINT "        Voraussichlich Fehler"
2750 PRINT "       bei der Funktionseingabe"
```

```
2760 SOUND 1,800,200,5
2770 FOR i=1 TO 5000:NEXT i
2780 RESUME 1350
2790 REM ***************************************************
```

Damit Sie den „3D-Funktionsplot" ausprobieren können, möchte ich einige Funktionen mit Parametern vorschlagen, die recht schöne Grafiken ergeben.

Funktion	xu	xo	yu	yo	zu	zo	xa	xe	za	ze	zw	xw
COS(Z)*X^2	−4	5	−8	15	0	10	−3	3	1	6	2	2
SIN(X)+ABS(X)	−5	5	−5	10	−5	10	−4	3	−4	8	4	4
COS(X)+COS(2*X)+COS(5*x)*Z												
	−1	6	−3	6	−1	6	−.5	5	−.5	2.4	2	2
	−2	2	−3	5	2	1.5	−1	1	−1	1	4	4
SIN(X*Z)	−1	6	−1	6	−1	6	0	4	0	4	4	2
SIN(X)*COS(Z)	−1	6	−1	6	−1	6	0	4	0	4	4	2
	−5	5	−3	3	−5	5	−4	4	−3	3	4	4
	−5	5	−5	10	−5	10	−4	3	−4	8	4	4
20*COS(SQR(X^2+Z^2))/(SQR(X^2+Z^2)+1												
	−20	20	−17	17	−15	15	−15	15	−15	13	2	2

(Ausdruck hierzu siehe Seite 119)

11.0 TEST und TESTR

Die Reihe der Grafikbefehle wird mit zwei Befehlen abgerundet, die testen, in welcher Farbe (Faßnummer) ein Punkt gesetzt ist. Der Befehl

TEST x, y

gibt die Faßnummer wieder, in der der Punkt an den Koordinaten 'x' und 'y' gesetzt ist. Ebenso wie bei PLOT, DRAW und MOVE ist auch eine relative Abfrage möglich, wobei 'x' und 'y' dann den Versatz von der aktuellen Grafikcursorposition angeben.

TESTR x,y

Mit Hilfe des TEST-Befehls ist es z.B. möglich, einen bestimmten Bereich des Bildschirms zu invertieren, indem geprüft wird, wie der Punkt gesetzt ist, um ihn dann entsprechend zu setzen oder zu löschen. Interessant scheint weiterhin die Anwendung für Spiele. Um z.B. ein „Raumschiff" abzuschießen, muß lokalisiert werden, ob die „Bombe" das „Raumschiff" trifft. Wurde die Farbe des „Raumschiffes" z.B. aus dem Farbfaß 1 gewählt, trifft die „Bombe" das „Raumschiff" immer dann, wenn die nächste Position der „Bombe" mit der Farbe aus Faß 1 gefüllt ist.

12.0 Die CENTRONICS-Schnittstelle

Die Schnittstelle zum Drucker ist beim CPC-464 standardmäßig als CENTRONICS-Schnittstelle ausgegelegt. Die Firma CENTRONICS führte bei ihren Produkten diese Schnittstelle ein, die heute als (inoffizielle) Norm gilt. Es handelt sich hierbei um eine Parallelschnittstelle. Neben den Datenleitungen existiert zusätzlich noch die STROBE- und die BUSY-Leitung, die kennzeichnen, ob Daten für den Drucker vorliegen und ob der Drucker empfangsbereit ist. Leider ist der Datenbus (Datenleitungen) beim CPC-464 unverständlicherweise nur mit 7 Leitungen ausgelegt worden, was zur Folge hat, daß nur Werte in der Größe 0-127 (x111 1111) dem Drucker übermittelt werden können. Bei Übermittlung der normalen ASCII-Zeichen (0-127) treten keinerlei Probleme auf. Allerdings besitzen die neueren Drucker wesentlich mehr Funktionen und Zeichen, die zum Teil nur über die ASCII-Codes 128-255 ansprechbar sind. Hierzu zählen vor allem die alternativen Zeichensätze, selbstdefinierbare Sonderzeichen und der „8-Bit-Modus" für die Grafikausgabe. Viele dieser Sonderfunktionen lassen sich auf Grund der fehlenden Datenleitung nicht oder nur mit erheblichem Programmieraufwand ansprechen.

12.1 Das Kabel zum Drucker

Für die Verbindung zwischen dem CPC-464 und dem Drucker benötigen Sie ein Kabel, das auf der einen Seite einen 34-poligen Platinenstecker und auf der anderen Seite einen CENTRONICS-Stecker besitzt. Falls Sie kein solches Kabel besitzen, können Sie sich dieses selber ohne großen Aufwand herstellen. Neben den besagten Steckern benötigen Sie noch ein Kabel mit mindestens 11 Leitungen (z.B. Flachbandkabel). Verbinden Sie dann die beiden Stecker entsprechend der folgenden Belegung:

CPC-464		CENTRONICS
1	STROBE	1
2	DATA 0	2
3	DATA 1	3
4	DATA 2	4
5	DATA 3	5
6	DATA 4	6
7	DATA 5	7

8	DATA 6	8
9	DATA 7	9
11	BUSY	11
16	GND	16

Die Kontaktbelegung des 34-poligen Druckeranschlussßes finden Sie im CPC-Handbuch (Anhang 5/Seite 2). Der CENTRONICS-Stecker ist mit Nummern bezeichnet.

12.2 PRINTen auf dem Drucker

Als Ausgabegerätenummer ist dem Drucker fest die Nummer 8 zugeordnet (die Nummern 0 bis 7 sind den Windows und die Nummer 9 der Kassette oder Diskette zugeordnet). Um ein Zeichen auf dem Drucker auszugeben, brauchen Sie keine weiteren Vorbereitungen zu treffen, außer natürlich den Drucker einzuschalten.

```
PRINT #8,''Druckertest''
```

Nach dem PRINT-Befehl folgt die Gerätenummer des Druckers (#8), dann mit einem Komma (',') getrennt der Ausgabetext. Die auf den Drucker bezogenen PRINT-Befehle verhalten sich ebenso wie auf dem Bildschirm. Wird als Trennzeichen hinter dem PRINT-Befehl ein Komma gesetzt, erscheint die nächste Ausgabe an der nächsten Tabulatorposition des Druckers.

```
10 FOR i=1 to 100
20 PRINT#8,"Hallo",
30 NEXT i
```

Auch alle anderen Funktionen wie TAB, SPC, USING etc. sind voll auf den Drucker übertragbar.
Das Listen eines Programms auf dem Drucker ist mit den Befehlen

LIST #8	Gesamtes Programm
LIST -1000, #8	Listen bis einschließlich Zeile 1000
LIST 1000-, #8	Ab einschließlich Zeile 1000
LIST 1000-2000, #8	Von Zeile 1000 bis Zeile 2000

möglich. Ein spezieller Drucker-Befehl bestimmt die Breite einer Zeile bei der Druckerausgabe.

```
WIDTH zeichen
```

Ist die Druckzeile länger als die mit „zeichen" bestimmte Anzahl, so wird der Druckkopf automatisch an den Anfang der nächsten Zeile positioniert. Ein Programmausdruck läßt sich somit in jeder gewünschten Breite (hier 40 Zeichen pro Zeile) ausdrucken.

```
WIDTH 40
LIST #8
```

12.3 Die Hardcopy

Unter einer Hardcopy versteht man den Ausdruck des momentanen Bildschirms auf dem Drucker. Besonders interessant ist das Ausdrucken von Grafiken, die z.B. mit dem 3-D Funktionsplot erstellt worden sind. Andererseits bietet sich auch ein reiner Textausdruck (z.B. einer Karteikarte) auf dem Drucker an.
Wie im Abschnitt zum Bildschirmspeicheraufbau (siehe 3.0) beschrieben, liegen alle Zeichen hier nur als gesetzte Punkte in den verschiedenen Farben vor. Die Information, um welches (ASCII-) Zeichen es sich handelt, ist nicht mehr vorhanden. Das bedeutet, daß jeder einzelne Punkt des Bildschirms auf dem Drucker ausgegeben werden muß. Es sind immerhin 128000 Punkte, die bearbeitet und dann an den Drucker weiterzuleiten sind.

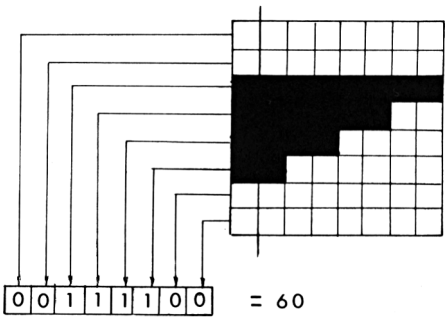
12.4 Grafikhardcopy

Als erstes stellt sich das Problem, wie festzustellen ist, ob ein Punkt gesetzt ist oder nicht. Im 80-Zeichenmodus (MODE 2) ist dieses noch recht leicht zu prüfen, da jedes gesetzte Bit des Bildschirmspeichers einem gesetzten Punkt auf dem Bildschirm entspricht (siehe 3.1). Anders ist dieses schon im 40- oder 20-Zeichenmodus (MODE 1 und MODE 0). Hier repräsentieren 2 oder 4 Bits jeweils einen Bildschirmpunkt (siehe 9.2). Auch kann es möglich sein, daß der Bildschirm gescrollt hat und sich daher der Bildschirmspeicher entsprechend verschoben hat. Ein direktes Auslesen des Bildschirmspeichers bietet sich also

aus den genannten Gründen nicht an. Wesentlich sinnvoller und vor allem einfacher ist die Nutzung des unter 11.0 beschriebenen TEST-Befehls. Mit Hilfe dieses Befehls läßt sich jede Position des Bildschirms auf seine Farbe hin abfragen. Ist z.B. der Bildschirmpunkt X = 100/Y = 100 mit der Farbe aus dem Faß Nummer 1 gefüllt, erhalten Sie mit

```
PRINT TEST 100,100
```

das Ergebnis „1". Somit ist es sehr einfach, den gesamten Bildschirm zu prüfen. Als nächstes Problem taucht die Frage auf, wie der Drucker diese Punkte ausgibt und wie sie ihm übermittelt werden müssen. Alle grafikfähigen Drucker besitzen den sogenannten „8-Bit-Druckmodus". Beim 8-Bit-Druckmodus wird ein Byte (8 Bits gleich 8 Punkte) senkrecht untereinander ausgedruckt, wobei ein gesetztes Bit als Punkt ausgegeben wird. Damit wird eine Umrechnung (wie aus der Zeichnung ersichtlich) notwendig:



Doch jetzt macht sich das fehlende 8.Bit am Druckerport störend bemerkbar. Die oberste Nadel (Punkt) des Druckers kann nicht mehr angesteuert werden. Als Lösung bietet sich an, weniger als 8 Bits zu übertragen und den Zeilenvorschub entsprechend zu verkleinern, so daß die nächste Punktreihe nahtlos an die obere Zeile gesetzt wird. Wählen Sie z.B. jeweils nur 4 Bildschirmpunkte aus (4 Bits = 1 Nibble) erhalten Sie auch nur Werte zwischen 0 und 15, die dem Drucker ohne Probleme übermittelt werden können (1111 = 15). Das folgende Grafikhardcopy-Programm ist in BASIC für die EPSON-Drucker geschrieben und arbeitet nach dem beschriebenen Prinzip. Alle EPSON kompatiblen Drucker, so auch der Schneiderdrucker, dürften mit diesem Programm zusammenarbeiten. In Zeile 65060 wird der Drucker normiert (Druckerreset) und in der Programmzeile 65070 wird der Zeilenvorschub auf 3/272 Zoll ein-

gestellt. Die Programmzeile 65090 aktiviert den 8-Bit-Modus und bestimmt, wieviele Daten pro Druckzeile folgen (639=2*256+127). Nacheinander prüft das Programm jeweils 4 untereinanderliegende Punkte und setzt, falls die Punktfarbe ungleich Null ist, das entsprechende Bit. Damit es zu keinen Verzerrungen bei der Y-Achse kommt, wird in der Y-Achse jeder Punkt doppelt abgefragt (400 Werte = 200 Punkte !!). Die ausgedruckte Hardcopy erhält somit das Format 640*400 Punkte und hat somit die gleichen Proportionen wie auf dem Bildschirm (Größe DIN A5).

```
65000 REM ************************************ M.W.Thoma **
65010 REM *                                               *
65020 REM *                    Grafikhardcopy             *
65030 REM *                    BASIC  Version             *
65040 REM *                                               *
65050 REM *****************************************************
65060 PRINT#8,CHR$(27);"§";
65070 PRINT#8,CHR$(27);"3";CHR$(12);
65080 FOR y=399+2 TO 2 STEP-4
65090 PRINT#8,CHR$(27);"*";CHR$(4);CHR$(127);CHR$(2);
65100 FOR x=0 TO 639:MOVE x,y
65110 FOR yr=3 TO 0 STEP-1
65120 IF TESTR (0,-1)=0 THEN 65140
65130 nibble=nibble+2^yr
65140 NEXT yr
65150 PRINT#8,CHR$(nibble);:nibble=0
65160 NEXT x
65170 PRINT#8,CHR$(13)
65180 NEXT y
65190 REM *****************************************************
```

Wesentlich schneller geht die Grafikhardcopy, wenn sie in Maschinensprache geschrieben ist (obwohl es hier auch noch einige Zeit dauert). Das vorliegende Maschinenprogramm arbeitet wie das BASIC-Programm. Nach jeder gedruckten Zeile können Sie die Hardcopy stoppen, indem Sie die TAB-Taste gedrückt halten. Nach dem BASIC-Ladeprogramm finden alle Maschinenspracheprogrammierer noch das komplette Assemblerlisting.

```
65000 REM ************************************ M.W.Thoma **
65010 REM *                                               *
65020 REM *        Grafikhardcopy  640 x 400 Punkte       *
65030 REM *           Aufruf mit  CALL &A000              *
65040 REM *                                               *
65050 REM *****************************************************
65060 MEMORY &9FFF
65070 FOR i= &A000 TO &A0A9
65080 READ d$:POKE i,VAL("&"+d$)
```

```
65090 summe=summe+VAL("&"+d$)
65100 NEXT i
65110 IF summe=19932 THEN RETURN
65120 PRINT"*** Fehler im Maschinenprogramm ***"
65130 STOP
65140 '
65150 DATA 3E,1B,CD,A1,A0,3E,40,CD,A1,A0,3E,1B,CD,A1,A0,3E
65160 DATA 33,CD,A1,A0,3E,0C,CD,A1,A0,CD,7C,A0,21,8F,01,22
65170 DATA AA,A0,11,00,00,2A,AA,A0,0E,00,06,04,E5,D5,C5,CD
65180 DATA F0,BB,C1,D1,E1,FE,00,37,20,01,3F,CB,11,2B,10,EC
65190 DATA 79,CD,A1,A0,13,21,80,02,37,ED,52,20,D8,CD,96,A0
65200 DATA 2A,AA,A0,37,01,02,00,ED,42,28,16,2A,AA,A0,2B,2B
65210 DATA 2B,2B,22,AA,A0,3E,44,CD,1E,BB,20,05,CD,7C,A0,18
65220 DATA B1,3E,1B,CD,A1,A0,3E,40,CD,A1,A0,C9,3E,1B,CD,A1
65230 DATA A0,3E,2A,CD,A1,A0,3E,04,CD,A1,A0,3E,7F,CD,A1,A0
65240 DATA 3E,02,CD,A1,A0,C9,3E,0D,CD,A1,A0,3E,0A,CD,A1,A0
65250 DATA C9,CD,2E,BD,38,FB,CD,31,BD,C9
65260 REM ***************************************************
```

```
                                ; ***************************************
                                ; *                                     *
                                ; *          GRAFIKHARDCOPY             *
                                ; *                                     *
                                ; *            CALL &A000               *
                                ; *                                     *
                                ; ***************************************
                                ;
A000                    ORG     &A000
A000            TEST    EQU     &BBF0
A000            BUSY    EQU     &BD2E
A000            PRTCHR  EQU     &BD31
A000
A000 3E1B               LD      A,27            ; DRUCKER NORMIEREN
A002 CDA1A0             CALL    PRINT
A005 3E40               LD      A,64
A007 CDA1A0             CALL    PRINT
A00A 3E1B               LD      A,27            ; ZEILENVORSCHUB AUF 12/216
A00C CDA1A0             CALL    PRINT
A00F 3E33               LD      A,51
A011 CDA1A0             CALL    PRINT
A014 3E0C               LD      A,12
A016 CDA1A0             CALL    PRINT
A019 CD7CA0             CALL    OPTION
A01C 218F01             LD      HL,399          ; Y = 399
A01F 22AAA0             LD      (HELP),HL       ; Y MERKEN
A022 110000     LAB4    LD      DE,0            ; X = 0
A025 2AAAA0     LAB3    LD      HL,(HELP)       ; LADE Y-POSITION
A028 0E00               LD      C,0             ; DRUCKMUSTER LOESCHEN
A02A 0604               LD      B,4             ; ZAEHLER
A02C E5         LAB2    PUSH    HL              ; MERKE ALLE REGISTER (STAPEL
A02D D5                 PUSH    DE
A02E C5                 PUSH    BC
```

```
A02F CDF0BB              CALL   TEST          ; TESTE X/Y-KOORDINATE
A032 C1                  POP    BC            ; HOLE ALLE REGISTER (STAPEL)
A033 D1                  POP    DE
A034 E1                  POP    HL
A035 FE00                CP     0             ; VERGLEICHE PUNKT MIT 0
A037 37                  SCF                  ; SETZE C-FLAG
A038 20FE                JR     NZ,LAB1       ; IST PUNKTFARBE <> 0 ?
A03A 3F                  CCF                  ; C-FLAG LOESCHEN
A03B CB11        LAB1    RL     C             ; C-FLAG IN <C> SCHIEBEN
A03D 2B                  DEC    HL            ; Y=Y-1
A03E 10EC                DJNZ   LAB2          ; IST <B> =0? NEIN ....
A040 79                  LD     A,C           ; ZEICHENMUSTER IN <AC>
A041 CDA1A0              CALL   PRINT         ; UND AUSDRUCKEN
A044 13                  INC    DE            ; X=X+1
A045 218002              LD     HL,640        ; PRUEFEN OB X=639
A048 37                  SCF
A049 ED52                SBC    HL,DE
A04B 20D8                JR     NZ,LAB3       ; X < 639 NAECHSTES ZEICHEN
A04D CD96A0              CALL   RETURN        ; NEUE ZEILE
A050 2AAAA0              LD     HL,(HELP)     ; PRUEFE OB LETZTE ZEILE
A053 37                  SCF
A054 010200              LD     BC,2
A057 ED42                SBC    HL,BC
A059 28FE                JR     Z,ENDE        ; JA, DANN ENDE
A05B 2AAAA0              LD     HL,(HELP)     ; NEIN
A05E 2B                  DEC    HL            ; Z-ACHSE UM 4 ERNIEDRIGEN
A05F 2B                  DEC    HL
A060 2B                  DEC    HL
A061 2B                  DEC    HL
A062 22AAA0              LD     (HELP),HL
A065 3E44                LD     A,68          ; PRUEFEN OB TAB-TASTE
A067 CD1EBB              CALL   &BB1E         ; GEDRUECKT IST ?
A06A 20FE                JR     NZ,ENDE       ; JA, DANN ENDE
A06C CD7CA0              CALL   OPTION        ; NEIN, 8-BIT-MODUS EINSTELLEN
A06F 18B1                JR     LAB4          ; UND WEITER MACHEN
A071
A071 3E1B        ENDE    LD     A,27          ; PROGRAMMENDE
A073 CDA1A0              CALL   PRINT         ; DRUCKER WIEDER NORMIEREN
A076 3E40                LD     A,64
A078 CDA1A0              CALL   PRINT
A07B C9                  RET
A07C
A07C 3E1B        OPTION  LD     A,27          ; SETZE 8-BIT-MODUS
A07E CDA1A0              CALL   PRINT
A081 3E2A                LD     A,42
A083 CDA1A0              CALL   PRINT
A086 3E04                LD     A,4
A088 CDA1A0              CALL   PRINT
A08B 3E7F                LD     A,127
A08D CDA1A0              CALL   PRINT
A090 3E02                LD     A,2
A092 CDA1A0              CALL   PRINT
A095 C9                  RET
A096
A096 3E0D        RETURN  LD     A,13          ; SENDE RETURN UND LINE FEED
```

```
A098 CDA1A0              CALL   PRINT
A09B 3E0A                LD     A,10
A09D CDA1A0              CALL   PRINT
A0A0 C9                  RET
A0A1
A0A1 CD2EBD      PRINT   CALL   BUSY          ; ZEICHEN IM <AC> DRUCKEN
A0A4 38FB                JR     C,PRINT
A0A6 CD31BD              CALL   PRTCHR
A0A9 C9                  RET
A0AA 00          HELP    DB     0
A0AB 00                  DB     0
A0AC
```

Bauen Sie sich das Grafikhardcopyprogramm überall dort ein, wo Sie eine Grafikhardcopy benötigen. Doch bitte beachten Sie dabei, daß sich die Punktabfrage in den Programmen immer auf das „normale" Grafiksystem bezieht. Wurde der Ursprung innerhalb des Programms geändert, müssen Sie vor Aufruf des Grafikhardcopyprogramms (CALL &A000) den Grafikbereich mit

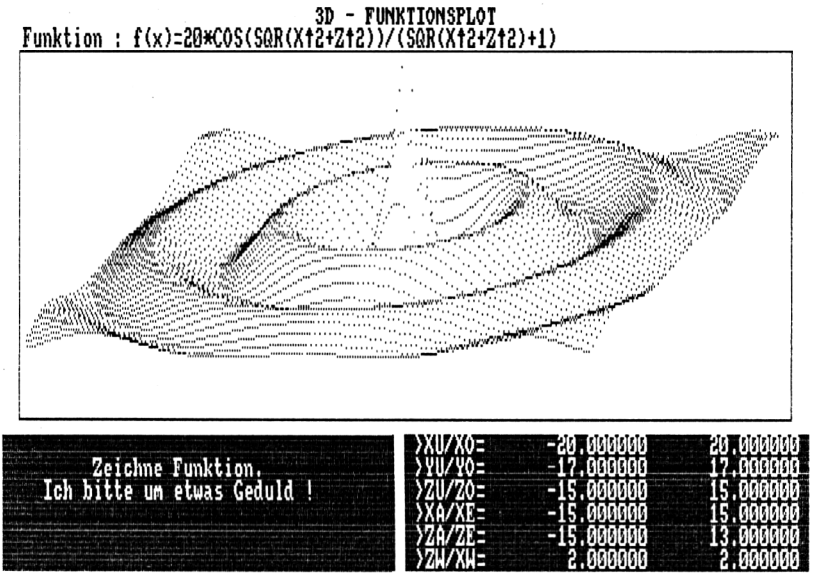
ORIGIN 0,0,0,639,0,399

normalisieren. Nach Durchführung der Hardcopy setzen Sie den Ursprung wieder in den ursprünglichen Zustand.

Erweitern Sie z.B. den 3-D-Funktionsplot um die Zeilen:

```
1415 IF UPPER$(in$)="D" THEN 3000

3000 CLS
3010 ORIGIN 0,0,0,639,0,399
3020 CALL &A000
3030 ORIGIN 14,106
3040 GOTO 1350
```

und um den BASIC-Lader für die Hardcopy ab Programmzeile 65000, so können Sie sich diesen besonders schönen Ausdruck (Originalgröße DIN A5) selber erstellen.



12.5 Die Texthardcopy

Die Grafikhardcopy ist immer nur dann sinnvoll, wenn auch tatsächlich Grafiken (Zeichnungen, Diagramme) vom Bildschirm ausgedruckt werden sollen. Befinden sich nur ASCII-Zeichen (32-127) auf dem Bildschirm, ist eine Grafikhardcopy schon auf Grund des nicht unerheblichen Zeitaufwandes nicht ratsam. Bei einer Texthardcopy ist nur zu prüfen, welches ASCII-Zeichen sich an einer bestimmten Position des Bildschirms befindet, um es dann entsprechend auf dem Drucker auszugeben. Als bekanntes Problem stellt sich wieder die Frage, wie läßt feststellen, welches Zeichen auf dem Bildschirm dargestellt ist? Eine Möglichkeit besteht darin, jeweils die 8*8 Punkte eines Zeichens mit dem Zeichen-ROM zu vergleichen. Sind alle 8 Bytes (64 Bit = 64 Punkte) identisch, ist ein gültiges ASCII-Zeichen gefunden. Aber ein BASIC-Programm braucht ca. 15 Minuten, um dieses Problem zu lösen. Bei solchen zeitkritischen Anwendungen bietet nur die Maschinensprache einen Ausweg, da sie Teile (Routinen) des vorhandenen Betriebssystems nutzen kann. Eine solche Routine ist die COPY-Tastenfunktion. Mit Hilfe der COPY-Taste können Sie einen Teil (Programmzeile etc) des Bildschirms an einen anderen Ort kopieren. Um diese Funktion durchzuführen, muß selbstverständlich festgestellt werden, welches Zeichen sich an der zu kopierenden Stelle befindet.

Handelt es sich um ein gültiges Zeichen, kann es kopiert werden. Falls nicht, gibt der CPC-464 eine „Fehlermeldung" in Form eines Tones aus. Eine dieser Routinen kann mit „CALL &BB60" angesprungen werden. Sie testet das Feld unter der aktuellen Cursorposition auf ein gültiges ASCII-Zeichen und legt es dann in den Akkumulator (zentrales Register im Z-80 Prozessor). Wird kein gültiges Zeichen gefunden, befindet sich im Akkumulator der Wert 0.

Das folgende Texthardcopyprogramm ist außer einem 7 Byte langen Maschinenprogramm vollständig in BASIC gehalten, so daß Sie es ohne Schwierigkeiten Ihren speziellen Wünschen anpassen können. Ohne weiteres lassen sich auch Ausschnitte des Bildschirms kopieren. Die Programmzeile 65080 bestimmt die Spaltenanzahl und ist dem jeweiligen Bildschirmmodus anzupassen (20, 40 oder 80 Zeichen). Das Maschinenprogramm besteht aus den Befehlen:

```
CALL &BB60      ;Prüfe Feld unter Cursor
LD (017C),A     ;und lege AC-Inhalt nach Adresse &017C.
RET             ;Springe zurück ins BASIC.
```

Da dieses Maschinenprogramm sehr kurz ist, kann es ohne Probleme in einer Remarkzeile untergebracht werden. Der Vorteil liegt darin, daß nicht ein Teil des BASIC-RAMs vor dem Überschreiben geschützt werden muß. Aus diesem Grunde müssen der Programmzeile 1 mindestens 8 Zeichen nach der REM-Anweisung folgen, denn hier liegt später das Maschinenprogramm und das gefundene Zeichen.

```
1 REM !!!!!!! <-- 8 Zeichen
9
65000 REM ************************************ M.W.Thoma **
65010 REM *                                              *
65020 REM *                Texthardcopy                  *
65030 REM *                                              *
65040 REM ************************************************
65050 FOR i=373 TO 379:READ x:POKE i,x:NEXT i
65060 DATA 205,96,187,50,124,1,201
65070 FOR z=1 TO 25:              'Zeilenanfang - Zeilenende
65080 FOR s=1 TO 80:              'Spaltenanfang-Spaltenende
65090 LOCATE s,z
65100 CALL 373
65110 x=PEEK(380)
65120 IF x<32 OR x>127 THEN x=32
65130 PRINT#8,CHR$(x);
65140 NEXT s
65150 PRINT#8,CHR$(13)
65160 NEXT z
65170 REM ************************************************
```

Für alle, denen die BASIC-Version der Texthardcopy noch zu lange dauert, folgt eine Lösung, die komplett in Maschinensprache geschrieben ist. Nachdem das Maschinenprogramm ab Adresse &A000 abgelegt ist, kann die Hardcopy mit CALL &A000 ausgeführt werden. Alle „Maschinensprache-Programmierer" finden im Anschluß wieder das komplette Assemblerlisting.

```
65000 REM ************************************ M.W.Thoma **
65010 REM *                                               *
65020 REM *                  Texthardcopy                 *
65030 REM *             Aufruf mit CALL &A000             *
65040 REM *                                               *
65050 REM *  POKE &A01C,Startzeile: POKE &A042,Endzeile+1  *
65060 REM *                                               *
65070 REM *************************************************
65080 MEMORY &9FFF
65090 FOR i= &A000 TO &A054
65100 READ d$:POKE i,VAL("&"+d$)
65110 summe=summe+VAL("&"+d$)
65120 NEXT i
65130 IF summe = 9103 THEN RETURN
65140 PRINT"*** Fehler im Maschinenprogramm ***"
65150 STOP
65160 '
65170 DATA CD,78,BB,22,55,A0,CD,11,BC,FE,00,28,08,FE
65180 DATA 01,28,08,06,50,18,06,06,14,18,02,06,28,2E
65190 DATA 01,26,01,E5,CD,75,BB,CD,60,BB,FE,00,20,02
65200 DATA 3E,20,CD,4C,A0,E1,24,7C,B8,20,EA,3E,0D,CD
65210 DATA 4C,A0,3E,0A,CD,4C,A0,2C,7D,FE,19,20,D8,2A
65220 DATA 55,A0,CD,75,BB,C9,CD,2E,BD,38,FB,CD,31,BD
65230 DATA C9
65240 REM *************************************************
```

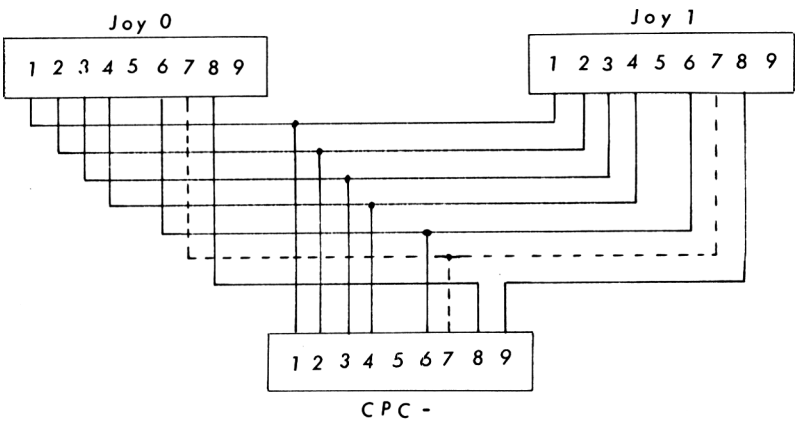
```
                        ; ********************************************
                        ; *                                          *
                        ; *              TEXTHARDCOPY                *
                        ; *                                          *
                        ; *               CALL &A000                 *
                        ; *                                          *
                        ; ********************************************
                        ;

A000            SETCUR  EQU    &BB75          ; SETZE CURSOR AUF <HL>
A000            GETCUR  EQU    &BB78          ; HOLE CURSORPOS. IN <HL>
A000            GETMOD  EQU    &BC11          ; HOLE MODE IM <AC>
A000            GETCHR  EQU    &BB60          ; ZEICHEN VON CURSORPOS. HOLEN
A000            PRTCHR  EQU    &BD31          ; ZEICHEN <AC> ZUM DRUCKER
A000            TESTPD  EQU    &BD2E          ; TESTE DRUCKBEREITSCHAFT
A000                                          ;
A000                    ORG    &A000
A000 CD78BB             CALL   GETCUR         ; HOLE ALTE CURSORPOS.
A003 2255A0             LD     (ALTCUR),HL    ; UND MERKE SIE.
```

```
A006 CD11BC              CALL   GETMOD        ; HOLE MODE
A009 FE00                CP     0             ; IST ES MODE 0 ?
A00B 28FE                JR     Z,MODE0       ; JA, DANN SPRINGE
A00D FE01                CP     1             ; IST ES MODE 1 ?
A00F 28FE                JR     Z,MODE1       ; JA, DANN SPRINGE
A011 0650                LD     B,80          ; ES IST MODE 2 (80 ZEICHEN)
A013 18FE                JR     START
A015 0614       MODE0    LD     B,20          ; ES IST MODE 0 (20 ZEICHEN)
A017 18FE                JR     START
A019 0628       MODE1    LD     B,40          ; ES IST MODE 1 (40 ZEICHEN)
A01B
A01B 2E01       START    LD     L,1           ; <L> IST DIE ZEILENPOS.
A01D 2601       LOOP2    LD     H,1           ; <H> IST DIE SPALTENPOS.
A01F E5         LOOP1    PUSH   HL            ; MERKE <HL> (STAPEL)
A020 CD75BB              CALL   SETCUR        ; SETZE CURSOR AUF ZEI/SPAL
A023 CD60BB              CALL   GETCHR        ; HOLE ZEICHEN VON CURSORPOS.
A026 FE00                CP     0             ; IST <AC> 0 (UNGUELTIG) ?
A028 20FE                JR     NZ,OK
A02A 3E20                LD     A,32          ; DANN GEBE SPACE AUS.
A02C CD4CA0     OK       CALL   PRINT         ; DRUCKE <AC> AUS.
A02F E1                  POP    HL            ; HOLE <HL> ZEILE/SPALTE
A030 24                  INC    H             ; ERHOEHE <H> SPALTE UM 1.
A031 7C                  LD     A,H           ; UND LADE ES IN DEN <AC>
A032 B8                  CP     B             ; ALLE SPALTE AUSGEDRUCKT ?
A033 20EA                JR     NZ,LOOP1      ; NEIN, NAECHSTE SPALTE.
A035 3E0D                LD     A,13          ; LADE <AC> MIT RETURN
A037 CD4CA0              CALL   PRINT         ; UND GEBE ES AUS.
A03A 3E0A                LD     A,10          ; LADE <AC> MIT LINE FEED
A03C CD4CA0              CALL   PRINT         ; UND GEBE ES ES AUS.
A03F 2C                  INC    L             ; ERHOEHE <L> ZEILE UM 1
A040 7D                  LD     A,L           ; UND LADE ES IN DEN <AC>
A041 FE19                CP     25            ; 25 ZEILEN AUSGEDRUCKT
A043 20D8                JR     NZ,LOOP2      ; NEIN, NAECHSTE ZEILE.
A045 2A55A0              LD     HL,(ALTCUR)   ; HOLE ALTE CURSORPOS. <HL>
A048 CD75BB              CALL   SETCUR        ; UND SETZE CURSOR.
A04B C9                  RET
A04C
A04C                                          ; DRUCKERAUSGABE
A04C CD2EBD     PRINT    CALL   TESTPD        ; DRUCKER BEREIT ?
A04F 38FB                JR     C,PRINT       ; NEIN, PRUEFE NEU.
A051 CD31BD              CALL   PRTCHR        ; GEBE ZEICHEN <AC> AUS
A054 C9                  RET
A055
A055 00         ALTCUR   DB     0
A056 00                  DB     0
```

13.0 Der Joystick

Der Joystick ist neben dem Paddle das bekannteste Mittel, um Spiele (spez. Actionspiele) einfach und rasant durchzuführen. Der Joystick ist heute bei fast jedem Spiel notwendig. Obwohl der CPC-464 nur einen Joystickeingang besitzt, lassen sich hier zwei Joysticks anschließen. Besitzen Sie einen schneidereigenen Joystick, so können Sie den zweiten Joystick an den ersten Joystick anschließen. Doch Sie brauchen sich nicht unbedingt den Schneider-Joystick anzuschaffen, um mit zwei Joysticks zu arbeiten. Ein kleiner Adapter genügt.

Der Joystick-Port ist eine Buchse mit 9 Kontakten, von denen allerdings nur 8 belegt sind (Stift 5 ist unbelegt). Die genaue Stiftbelegung entnehmen Sie bitte dem CPC-Handbuch (Anhang 5/Seite 1). Den Stiften ist jeweils eine Richtung sowie die beiden Feuertasten zugeordnet. Über die COMMON-Leitung (Stift 8) wird der erste Joystick selektiert. Für eine gewisse Zeit legt das Betriebssystem eine Spannung (+5V) auf die COMMON-Leitung. Ist in dieser Zeit der erste Joystick betätigt (in eine Richtung gedrückt oder eine Feuertaste betätigt), wird die Spannung auf einen entsprechenden Stift gelegt. Der zweite Joystick kann über die Leitung COM 2 aktiviert werden. COM 2 wird immer dann auf +5V gelegt, wenn COMMON passiv (0V) ist (und umgekehrt). Nun kann auf den gleichen Leitungen der zweite Joystick abgefragt werden.
Der Adapter besteht aus zwei 9 poligen Stiftsteckern und einem 9 poligen Buchsenstecker, die entsprechend der Schaltung zu verdrahten sind.



13.1 JOY - Abfrage der Joysticks

Mit dem Befehl

JOY (nummer)

läßt sich sehr einfach die momentane Stellung der Joysticks abfragen. Die „nummer" bestimmt, welcher Joystick abzufragen ist.

JOY (0)	1.Joystick (COMMON)
JOY (1)	2.Joystick (COM 2)

Die Abfrage der Joysticks liefert einen numerischen Ausdruck, wobei jede mögliche Bewegungsrichtung durch ein gesetztes Bit repräsentiert wird.

00000000	=	0	keine Betätigung (Ruhestellung)
00000001	=	1	Oben
00000010	=	2	Unten
00000100	=	4	Links
00001000	=	8	Rechts
00010000	=	16	Feuer 1
00100000	=	32	Feuer 2

Wollen Sie zum Beispiel abfragen, ob der erste Joystick (JOY (0)) nach oben gedrückt ist, so brauchen Sie nur den Joystickwert mit einer 1 zu vergleichen:

```
10 IF JOY(0) = 1 THEN PRINT ''Oben''
```

Ist der Joystick in einer diagonalen Richtung gedrückt (z.B. oben-links), so sind beide „Hauptrichtungs-Bits", also Bit 0 und Bit 2 gesetzt.

0000 0101 = 1+4 = 5 Oben-Links

Die Joystickabfrage liefert den dezimalen Wert 5. Alle anderen Kombinationen ergeben sich entsprechend:

0000 1010	=	2+8	= 10	Unten-rechts
0001 0101	=	16+4+1	= 21	Oben-links-Feuer 1

Mit Hilfe einer AND-Verknüpfung (siehe 20.2) läßt sich der Zustand jedes Bits einzeln abfragen.

IF (JOY(0) AND 2^bit) = 2^bit THEN .... Bedingung erfüllt

Die Variable „bit" bestimmt das zu prüfende Bit (0 bis 7). Ein konkretes Zahlenbeispiel soll die Vorgehensweise verdeutlichen:

JOY	: 0000 1000	= 8	= 2^3
AND 2^3 :	0000 1000	= 8	
Ergebnis	: 0000 1000	= 8	= 2^3 Bedingung erfüllt

JOY	: 0000 0100	= 4	= 2^2
AND 2^3	: 0000 1000	= 8	
Ergebnis	: 0000 0000	= 0	Bedingung nicht erfüllt

Abschließend zeigt das Demonstrationsprogramm noch einmal die Zusammenhänge zwischen Bit und Richtung.

```
1000 REM ***************************************************
1010 REM *                                                 *
1020 REM *              Joystick-Demonstration             *
1030 REM *                                                 *
1040 REM ***************************************************
1050 MODE 1
1060 PRINT "  Bitte Joystick (0) anschliessen !!"
1070 PRINT
1080 x=JOY(0):IF x=0 THEN 1080
1090 PRINT "XX";BIN$(x,6),
1100 IF (x AND 2^0)=2^0 THEN PRINT"Oben-";
1110 IF (x AND 2^1)=2^1 THEN PRINT"Unten-";
1120 IF (x AND 2^2)=2^2 THEN PRINT"Links-";
1130 IF (x AND 2^3)=2^3 THEN PRINT"Rechts-";
1140 IF (x AND 2^4)=2^4 THEN PRINT"Feuer1-";
1150 IF (x AND 2^5)=2^5 THEN PRINT"Feuer2-";
1160 PRINT CHR$(8);" "
1170 GOTO 1080
1180 REM ***************************************************
```

14.0 ELOCAD - ein Zeichenprogramm

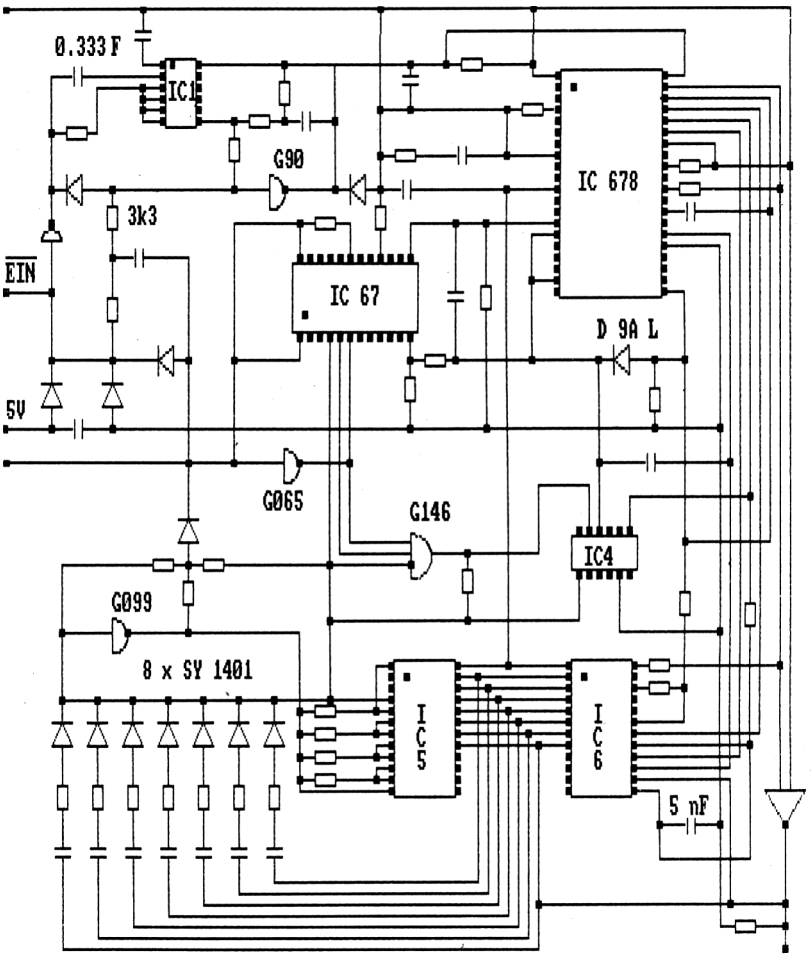
Das Schlagwort CAD (Computer Aided Design, Computer unterstützter Entwurf) steht für Computergrafik in Vollendung. Mit CAD-Systemen kann heute jedes erdenkliche Teil am Computer entwickelt und anschließend zu Papier gebracht werden (Ausdruck). Das hauptsächliche Anwendungsgebiet liegt im Entwicklungs- und Ingenieurwesen. CAD-Systeme stellen sehr hohe Anforderungen an die Bildschirmauflösung sowie an die Rechen- und Verarbeitungsgeschwindigkeit. Mit dem CPC-464 lassen sich natürlich solche Anwendungen nicht realisieren. Allerdings ist es durchaus möglich (im Rahmen der gegebenen technischen Möglichkeiten) mit dem CPC-464 Zeichnungen, Schaltungen etc. zu entwickeln. Besonders interessant erscheint mir die Möglichkeit, elektronische Schaltbilder mit dem Computer zu entwerfen und auf einem Drucker auszugeben.

Einen solchen elektronischen „Malkasten" möchte ich Ihnen auf den nächsten Seiten vorstellen. Er ist zum Erstellen von Schaltbildern ausgelegt worden. Sie können aus einer Palette von vordefinierten Schaltsymbolen auswählen und Sie an eine beliebige Stelle des Bildschirms setzen. Mit Linien lassen sich dann alle Anschlüsse verbinden und es entsteht ein Schaltbild. Ein mögliches Ergebnis zeigt die folgende „Schaltung" (die Schaltung ergibt keinen Sinn, bitte nicht nachbauen!!). Sie ist mit dem Programm „ELOCAD" erstellt und ausgedruckt worden (Original Größe DIN A4).

Das Programm ELOCAD erlaubt es, mit Hilfe des Joysticks (JOY 0) Dioden, Widerstände, Kondensatoren, IC's, Gatter, Linien, Texte und Punkte auf den Bildschirm zu zeichnen. Die erstellten Schaltpläne können auf Diskette oder Kassette gespeichert und von ihnen wieder geladen werden. Es ist auch möglich, mehrere Teilpläne hintereinander zu drucken, wodurch „unendlich" große Schaltpläne entstehen können. Das Programm kann (und soll) ohne weiteres weiter ausgebaut werden. Mit dem Joystick ist der Cursor (ein kleines Quadrat) frei auf dem Bildschirm bewegbar. Nach Betätigung der Feuertaste (Feuer 1) fragt das Programm nach einem Befehl. Es stehen Ihnen folgende Möglichkeiten zur Verfügung:

P : Punkt in Größe des Cursors
R : Widerstand waagerecht/senkrecht
D : Diode waagerecht/senkrecht
K : Kondensator waagerecht/senkrecht

I : IC waagerecht/senkrecht
G : Gatter waagerecht/senkrecht
T : Text einfügen
L : Linie von/nach
O : Zeichnung ausdrucken
C : Teile des Bildes löschen
1 : Bild abspeichern
2 : Bild laden
3 : Bild löschen



Alle Schaltzeichen werden an die aktuelle Cursorposition gesetzt. Beim Drucken des Schaltplanes werden die unteren 16 Punktlinien nicht mit ausgedruckt. Sie bilden den „Übertrag" zur nächsten Teilzeichnung. Löschen Sie den Bildschirm mit „3", so „scrollt" der Bildschirm nach oben, bis die unteren 16 Punktlinien oben angekommen sind. So lassen sich Leitungen problemlos auf die nächste Teilschaltung übertragen.

Versuchen Sie das Programm weiter auszubauen (mehr Schaltzeichen) und zu perfektionieren.

```
1000 REM ************************************ M.W.Thoma **
1010 REM *                                               *
1020 REM *                    E L O C A D                *
1030 REM *                                               *
1040 REM ****************************************************
1050 MODE 2
1060 DIM cr(9),ma(9)
1070 INK 0,0:INK 1,26:BORDER 0
1080 WINDOW#0,1,80,23,25
1090 WINDOW#1,1,80,1,22
1100 ORIGIN 0,48,0,639,48,399
1110 PEN#0,0:PAPER#0,1:CLS
1120 PEN#1,1:PAPER#1,0
1130 x=10:y=10:GOSUB 3520
1140 DEG
1150 GOSUB 65000:POKE &A055,66:POKE &A056,0
1160 '
1170 REM ****************************************************
1180 REM *                                                  *
1190 REM *          Joystick- und Tastaturabfrage           *
1200 REM *                                                  *
1210 REM ****************************************************
1220 IF (JOY(0)=1 AND y<344)THEN GOSUB 3680:y=y+8:GOSUB 3520
1230 IF (JOY(0)=2 AND y>0)  THEN GOSUB 3680:y=y-8:GOSUB 3520
1240 IF (JOY(0)=4 AND x>0)  THEN GOSUB 3680:x=x-4:GOSUB 3520
1250 IF (JOY(0)=8 AND x<639)THEN GOSUB 3680:x=x+4:GOSUB 3520
1260 IF JOY(0)=16 THEN 1290
1270 GOTO 1170
1280 '
1290 IF mark=1 THEN mark=0:GOTO 1740
1300 IF loesch=1 THEN loesch=0:FOR t=1 TO 200:NEXT:GOTO 1500
1310 CLS:PRINT "  > Befehl  ?"
1320 bf$=INKEY$:IF bf$="" THEN 1320
1330 IF bf$=CHR$(13) THEN 1500
1340 IF UPPER$(bf$)="P" THEN 1520
1350 IF UPPER$(bf$)="L" THEN 1640
1360 IF UPPER$(bf$)="R" THEN 1860
1370 IF UPPER$(bf$)="I" THEN 2050
1380 IF UPPER$(bf$)="G" THEN 2380
1390 IF UPPER$(bf$)="T" THEN 2630
1400 IF UPPER$(bf$)="O" THEN 2730
1410 IF UPPER$(bf$)="K" THEN 2860
```

```
1420 IF UPPER$(bf$)="C" THEN 3040
1430 IF UPPER$(bf$)="D" THEN 3140
1440 IF UPPER$(bf$)="Q" THEN 3340
1450 IF UPPER$(bf$)="1" THEN 3770
1460 IF UPPER$(bf$)="2" THEN 3890
1470 IF UPPER$(bf$)="3" THEN 4000
1480 GOTO 1320
1490 '
1500 CLS:GOTO 1170
1510 '
1520 REM ***************************************************
1530 REM *                                                 *
1540 REM * <P>          Verbindungspunkt setzen            *
1550 REM *                                                 *
1560 REM ***************************************************
1570 FOR i=1 TO 9: ct(i)=1:NEXT i
1580 MOVER -2,2
1590 FOR i=0 TO 4 STEP 2
1600 DRAWR 4,0,1:MOVER -4,-2
1610 NEXT i
1620 GOTO 1500
1630 '
1640 REM ***************************************************
1650 REM *                                                 *
1660 REM * <L>          Linie zeichnen von/nach            *
1670 REM *                                                 *
1680 REM ***************************************************
1690 CLS:PRINT "  Linie zeichnen von hier nach .... "
1700 x11=x:y11=y
1710 FOR i=1 TO 9:ma(i)=ct(i):ct(i)=1:NEXT i
1720 mark=1:GOTO 1170
1730 '
1740 x12=x:y12=y
1750 IF x11=x12 AND y11=y12 THEN 1830
1760 MOVE x11,y11
1770 PLOTR -2,2,ma(1):PLOTR 2,0,ma(2):PLOTR 2,0,ma(3)
1780 PLOTR 0,-2,ma(4):PLOTR 0,-2,ma(5):PLOTR -2,0,ma(6)
1790 PLOTR -2,0,ma(7):PLOTR 0,2,ma(8):PLOTR 2,0,ma(9)
1800 MOVE x11,y11:DRAW x12,y12,1
1810 FOR t=1 TO 200:NEXT t:GOTO 1500
1820 '
1830 FOR i=1 TO 9:ct(i)=ma(i):NEXT i
1840 FOR t=1 TO 200:NEXT t:GOTO 1500
1850 '
1860 REM ***************************************************
1870 REM *                                                 *
1880 REM * <R>          Widerstaende setzen                *
1890 REM *                                                 *
1900 REM ***************************************************
1910 CLS:PRINT "  Widerstand  (s)enkrecht / (w)aagerecht  ?"
1920 b$=INKEY$:IF b$="" THEN 1920
1930 IF UPPER$(b$)="S" THEN 1970
1940 IF UPPER$(b$)="W" THEN 2010
1950 GOTO 1500
1960 '
```

```
1970 MOVER 0,-4:DRAWR 0,-8,1:MOVER 0,-16:DRAWR 0,-8,1
1980 MOVER -4,24:yr=16:xr=8:GOSUB 3440
1990 GOTO 1500
2000 '
2010 MOVER 4,0:DRAWR 8,0,1:MOVER 16,0:DRAWR 8,0,1
2020 MOVER -24,4:xr=16:yr=8:GOSUB 3440
2030 GOTO 1500
2040 '
2050 REM ***************************************************
2060 REM *                                                 *
2070 REM * <I>            IC - zeichnen                    *
2080 REM *                                                 *
2090 REM ***************************************************
2100 CLS:PRINT "  IC   (s)enkrecht / (w)aagerecht  ?"
2110 b$=INKEY$:IF b$="" THEN 2110
2120 IF (UPPER$(b$)="S") OR (UPPER$(b$)="W") THEN 2140
2130 GOTO 1500
2140 CLS:INPUT "  Wieviele Pins :";pin
2150 IF pin/2 <> INT(pin/2) THEN 2140
2160 pin=pin/2
2170 IF UPPER$(b$)="W" THEN 2270
2180 xr=pin*8/2:yr=pin*8:MOVER 2,-3:GOSUB 3440
2190 FOR j=1 TO pin
2200 FOR i=0 TO 4
2210 MOVE x-2,y-j*8+i-2:DRAWR 4,0,1
2220 MOVER pin/2*8,0:DRAWR 4,0,1
2230 NEXT i
2240 NEXT j
2250 GOTO 1500
2260 '
2270 xr=pin*8+4
2280 yr=pin/2*8
2290 MOVER 2,-3:GOSUB 3440
2300 FOR j=1 TO pin
2310 FOR i=0 TO 4
2320 MOVE x+j*8+i-2,y-2:DRAWR 0,4,1
2330 MOVER 0,-pin/2*8-8:DRAWR 0,-4,1
2340 NEXT i
2350 NEXT j
2360 GOTO 1500
2370 '
2380 REM ***************************************************
2390 REM *                                                 *
2400 REM * <G>            Gatter zeichnen                  *
2410 REM *                                                 *
2420 REM ***************************************************
2430 CLS:PRINT "  Gatter  (s)enkrecht / (w)aagerecht  ?"
2440 b$=INKEY$:IF b$="" THEN 2440
2450 IF UPPER$(b$)="S" OR UPPER$(b$)="W" THEN 2470
2460 GOTO 1500
2470 INPUT "  Wieviele Eingaenge :";eg
2480 IF eg<1 OR eg>8 THEN 1500
2490 IF eg=1 THEN eg=2
2500 IF UPPER$(b$)="W" THEN wa=180:we=360::GOTO 2530
2510 wa=270:we=450:eg=eg+1
```

```
2520 x1=x:y1=y:x=x-4*eg-4:y=y+4*eg+2
2530 FOR w=wa TO we STEP 4
2540 PLOT x+eg*4*COS(w),(y-4-eg*4)-eg*4*SIN(w),1
2550 NEXT w
2560 IF UPPER$(b$)="W" THEN 2600
2570 DRAWR 0,8*eg,1:x=x1:y=y1
2580 GOTO 1500
2590 '
2600 DRAWR -8*eg,0,1
2610 GOTO 1500
2620 '
2630 REM ***************************************************
2640 REM *                                                 *
2650 REM * <T>            Text einfuegen                   *
2660 REM *                                                 *
2670 REM ***************************************************
2680 CLS:INPUT "Text einfuegen :";tx$
2690 IF tx$="" THEN 1500
2700 PLOT -10,-10,1:MOVE x,y-4:TAG:PRINT tx$;:TAGOFF
2710 GOTO 1500
2720 '
2730 REM ***************************************************
2740 REM *                                                 *
2750 REM * <O>        Zeichnung ausdrucken                 *
2760 REM *                                                 *
2770 REM ***************************************************
2780 CLS:PRINT TAB(12);"Manfred Walter Thoma        ";
2790 PRINT "E L O C A D    ";CHR$(164);" 1985    Hamburg"
2800 PRINT "  Zeichnung ausdrucken .... bitte warten"
2810 ORIGIN 0,0,0,639,0,399
2820 CALL &A000
2830 ORIGIN 0,48,0,639,48,399
2840 GOTO 1500
2850 '
2860 REM ***************************************************
2870 REM *                                                 *
2880 REM * <K>        Kondensator zeichnen                 *
2890 REM *                                                 *
2900 REM ***************************************************
2910 CLS:PRINT" Kondensator   (s)enkrecht / (w)aagerecht  ?"
2920 b$=INKEY$:IF b$="" THEN 2920
2930 IF UPPER$(b$)="S" THEN 2960
2940 IF UPPER$(b$)="W" THEN 3000
2950 GOTO 1500
2960 MOVER 0,-4:DRAWR 0,-12,1:MOVER 0,-8:DRAWR 0,-12,1
2970 MOVER -6,20:DRAWR 12,0,1:MOVER  0,-6:DRAWR -12,0,1
2980 GOTO 1500
2990 '
3000 MOVER 4,0:DRAWR 12,0,1:MOVER 6,0:DRAWR 12,0,1
3010 MOVER -12,6:DRAWR 0,-12,1:MOVER  -6,0::DRAWR 0,12,1
3020 GOTO 1500
3030 '
3040 REM ***************************************************
3050 REM *                                                 *
3060 REM * <C>          Loeschen von Punkten               *
```

```
3070 REM *                                                    *
3080 REM ***************************************************
3090 CLS:PRINT "  Loeschen von Punkten "
3100 FOR i=1 TO 9:ct(i)=0:NEXT
3110 loesch=1
3120 GOTO 1170
3130 '
3140 REM ***************************************************
3150 REM *                                                    *
3160 REM * <D>              Diode zeichnen                   *
3170 REM *                                                    *
3180 REM ***************************************************
3190 CLS:PRINT" Diode   (s)enkrecht / (w)aagerecht  ?"
3200 b$=INKEY$:IF b$="" THEN 3200
3210 IF UPPER$(b$)="S" THEN 3240
3220 IF UPPER$(b$)="W" THEN 3290
3230 GOTO 1500
3240 DRAWR 0,-14,1:MOVER 0,-20:DRAWR 0,-12,1
3250 MOVER -8,30:DRAWR 16,0,1:MOVER -8,0:DRAWR -8,-16,1
3260 DRAWR 16,0,1:DRAWR -8,16,1
3270 GOTO 1500
3280 '
3290 DRAWR 12,0,1:MOVER 12,0:DRAWR 10,0,1
3300 MOVER -22,-10:DRAWR 0,20,1:MOVER 0,-10:DRAWR 12,10,1
3310 DRAWR 0,-20,1:DRAWR -12,10,1
3320 GOTO 1500
3330 '
3340 REM ***************************************************
3350 REM *                                                    *
3360 REM * <Q>              Programmende                     *
3370 REM *                                                    *
3380 REM ***************************************************
3390 CLS:PRINT"  Programmende !!! Sind Sie sicher ???"
3400 b$=INKEY$:IF b$="" THEN 3400
3410 IF UPPER$(b$)="J" THEN MODE 2:CLS:END
3420 GOTO 1500
3430 '
3440 REM ***************************************************
3450 REM *                                                    *
3460 REM *               Rechteck zeichnen                    *
3470 REM *                                                    *
3480 REM ***************************************************
3490 DRAWR xr,0,3:DRAWR 0,-yr,3:DRAWR -xr,0,3:DRAWR 0,yr,3
3500 RETURN
3510 '
3520 REM ***************************************************
3530 REM *                                                    *
3540 REM *           Grafikcursor setzen/loeschen             *
3550 REM *                                                    *
3560 REM ***************************************************
3570 MOVE x,y
3580 ct(1)=TESTR(-2,2):ct(2)=TESTR(2,0):ct(3)=TESTR(2,0)
3590 ct(4)=TESTR(0,-2):ct(5)=TESTR(0,-2):ct(6)=TESTR(-2,0)
3600 ct(7)=TESTR(-2,0):ct(8)=TESTR(0,2):ct(9)=TESTR(2,0)
3610 MOVE x,y
```

```
3620 PLOTR -2,2,1:PLOTR 2,0,1:PLOTR 2,0,1
3630 PLOTR 0,-2,1:PLOTR  0,-2,1:PLOTR -2,0,1
3640 PLOTR -2,0,1:PLOTR 0,2,1:PLOTR 2,0,0
3650 MOVE x,y
3660 RETURN
3670 '
3680 MOVE x,y
3690 IF loesch=1 THEN FOR i=1 TO 9:ct(i)=0:NEXT i
3700 PLOTR -2,2,ct(1):PLOTR 2,0,ct(2):PLOTR 2,0,ct(3)
3710 PLOTR 0,-2,ct(4):PLOTR 0,-2,ct(5):PLOTR -2,0,ct(6)
3720 PLOTR -2,0,ct(7):PLOTR 0,2,ct(8):PLOTR 2,0,ct(9)
3730 MOVE x,y:IF loesch=0 THEN 3750
3740 FOR i=0 TO 6:MOVE x+i,y+2:DRAWR 0,-6,0:NEXT:MOVE x,y
3750 RETURN
3760 '
3770 REM ***************************************************
3780 REM *                                                 *
3790 REM * <1>            Bild abspeichern                 *
3800 REM *                                                 *
3810 REM ***************************************************
3820 CLS:INPUT "  Speichern  Name (6 Zch) :";na$
3830 IF na$="" THEN 1500
3840 IF LEN(na$)>6 THEN 3820
3850 CLS:PRINT "  Zeichnung :";na$
3860 SAVE "!"+na$+".bin",b,&C000,&4000
3870 GOTO 1500
3880 '
3890 REM ***************************************************
3900 REM *                                                 *
3910 REM * <2>            Bild einladen                    *
3920 REM *                                                 *
3930 REM ***************************************************
3940 CLS:INPUT "  Laden  Name (6 Zch) :";na$
3950 IF na$="" THEN 1500
3960 IF LEN(na$)>6 THEN 3940
3970 LOAD "!"+na$+".bin",&C000
3980 GOTO 1500
3990 '
4000 REM ***************************************************
4010 REM *                                                 *
4020 REM * <3>             Bild loeschen                   *
4030 REM *                                                 *
4040 REM ***************************************************
4050 CLS:PRINT "  Bild loeschen   sicher ??????"
4060 b$=INKEY$:IF b$="" THEN 4060
4070 IF UPPER$(b$)="J" THEN 4090
4080 GOTO 1500
4090 WINDOW SWAP 0,1:PAPER 0:LOCATE 1,1
4100 FOR i=1 TO 43:PRINT:NEXT
4110 WINDOW SWAP 0,1:PAPER 1
4120 GOTO 1500
4130 '
65000 REM ************************************ M.W.Thoma **
65010 REM *                                               *
65020 REM *       Grafikhardcopy  640 x 400 Punkte        *
```

```
65030 REM *            Aufruf mit  CALL &A000                 *
65040 REM *                                                   *
65050 REM *************************************************
65060 MEMORY &9FFF
65070 FOR i= &A000 TO &A0A9
65080 READ d$:POKE i,VAL("&"+d$)
65090 summe=summe+VAL("&"+d$)
65100 NEXT i
65110 IF summe=19932 THEN RETURN
65120 PRINT"*** Fehler im Maschinenprogramm ***"
65130 STOP
65140 '
65150 DATA 3E,1B,CD,A1,A0,3E,40,CD,A1,A0,3E,1B,CD,A1,A0,3E
65160 DATA 33,CD,A1,A0,3E,0C,CD,A1,A0,CD,7C,A0,21,8F,01,22
65170 DATA AA,A0,11,00,00,2A,AA,A0,0E,00,06,04,E5,D5,C5,CD
65180 DATA F0,BB,C1,D1,E1,FE,00,37,20,01,3F,CB,11,2B,10,EC
65190 DATA 79,CD,A1,A0,13,21,80,02,37,ED,52,20,D8,CD,96,A0
65200 DATA 2A,AA,A0,37,01,02,00,ED,42,28,16,2A,AA,A0,2B,2B
65210 DATA 2B,2B,22,AA,A0,3E,44,CD,1E,BB,20,05,CD,7C,A0,18
65220 DATA B1,3E,1B,CD,A1,A0,3E,40,CD,A1,A0,C9,3E,1B,CD,A1
65230 DATA A0,3E,2A,CD,A1,A0,3E,04,CD,A1,A0,3E,7F,CD,A1,A0
65240 DATA 3E,02,CD,A1,A0,C9,3E,0D,CD,A1,A0,3E,0A,CD,A1,A0
65250 DATA C9,CD,2E,BD,38,FB,CD,31,BD,C9
65260 REM *************************************************
```

15.0 Tastatur

Die Tastatur des CPC-464 ist sehr großzügig ausgestattet. Sie ist in 3 Teile gegliedert:

1) alpha-numerische Tastatur (66 Tasten)
2) 10'er Block (12 Tasten)
3) Cursortasten mit COPY-Taste (5 Tasten)

Jede dieser 74 Tasten ist einer Nummer zugeordnet, die Sie aus dem Anhang des CPC-Handbuchs (Anhang 3/Seite 16) entnehmen können. Die beiden SHIFT-Tasten liegen parallel und besitzen daher die gemeinsame Tastennummer 21. Den beiden Joysticks sind ebenfalls für jede mögliche Stellung Tastennummern zugeordnet (48 bis 53 und 72 bis 77). Die Nummerierung beginnt bei 0 für die „Cursor up"- und endet mit 79 für die „DELete"-Taste.

Diesen Tastennummern sind mit Ausnahme der ESCAPE-, SHIFT-, CONTROL- und CAPS LOCK-Taste mehrere ASCII-Zeichen zugeordnet. Die Belegung der Tasten mit ASCII-Zeichen entnehmen Sie bitte dem CPC-464-Handbuch (Anhang 3/Seite 14), wobei die ASCII-Werte als hexadezimale Zahlen angegeben sind. Insgesamt ist es möglich, alle Tasten dreifach zu belegen:

1) nur die Taste betätigen
2) gleichzeitiges Betätigen von SHIFT und der Taste
3) gleichzeitiges Betätigen von CRTL und der Taste

Bei der Tastenbelegung im Handbuch entspricht die obere Zahl den ASCII-Zeichen von CRTL / Taste, die mittlere von SHIFT / Taste und die untere Zahl gilt, wenn nur die Taste gedrückt wird. Sehen Sie sich z.B. die Taste Nummer 65 einmal genauer an. Die Taste mit der Nummer 65 ist die „2" oben links auf der Tastatur (nicht auf dem 10'er Block). Bei Betätigung der Taste erscheint das Zeichen

ASCII Hex 32 = 50 = „2"

In Kombination mit der SHIFT-Taste erscheint der „Anführungsstrich"

ASCII HEX 22 = 34 = „"

und bei Kombination mit der CRTL-Taste ein Grafikzeichen (gerasterter Strich oben). Alle Tasten (mit Ausnahme der oben genannten), bei denen kein Zeichen für die Kombination CRTL/Taste angeben ist, sind mit dem ASCII-Wert „0" belegt. Das ASCII-Zeichen 0 wird ignoriert und hat keinerlei Einfluß.

Bis auf die Tasten „TAB", „ESC" und „CAPS LOCK", sowie dem gesamten 10'er Block besitzen alle im Einschaltzustand eine sogenannte Dauerfunktion (repeat), die bei längerer Betätigung eingeschaltet wird.

15.1 KEY DEF - Neubelegung der Tasten

Sämtlichen Tasten, mit Ausnahmen der SHIFT- und CRTL-Taste, können anderen ASCII-Zeichen zugeordnet werden. Dadurch ist es möglich, Tasten zu vertauschen oder aber Grafikzeichen auf die Tastatur zu legen. Mit Hilfe des Befehls

```
KEY DEF tastennr, repeat, normal, shift, crtl
```

kann der Taste mit der Nummer „tastennr" ein neues Zeichen zugeordnet werden. „normal", „shift" und „crtl" bestimmen die ASCII-Zeichen, welche ab sofort bei Betätigen der jeweiligen Kombination auf dem Bildschirm erscheinen. „repeat" kann einen Wert von 0 oder 1 annehmen, wobei mit einer 1 die Dauerfunktion für diese Taste aktiv ist. Als Beispiel soll die Belegung der „Z"- und „Y"-Taste miteinander vertauscht werden:

```
„Z" = Tastennummer 71
„Y" = Tastennummer 43

KEY DEF 71,1,&79,&59,&19
KEY DEF 43,1,&7A,&5A,&1A
```

Die ASCII-Werte (hier hexadezimale Zahlen) sind dem Tastaturbelegungsplan aus dem CPC-Handbuch entnommen (Anhang 3/Seite 14) und entsprechend getauscht worden.
Werden weniger Parameter eingegeben,

```
KEY DEF 71,1,&24
```

bleiben alle nicht definierten Kombinationen (hier SHIFT und CRTL) unverändert. Auch die normalerweise über die Tastatur nicht erreichbaren Grafikzeichen (Symbole) können so auf die verschiedenen Tasten gelegt werden. Mit CRTL/SPACE soll das Copyright-Zeichen erreichbar sein:

```
KEY DEF 47,1,&20,&20,&A4
```

Eine besondere Aufgabe besitzen die ASCII-Zeichen 128 bis 159 (&80 bis &9F). Versuchen Sie einmal das ASCII-Zeichen 144 (&90) auf die „Z"-Taste zu legen.

```
KEY DEF 71,1,&90
```

Betätigen Sie jetzt die „Z"-Taste, so erscheint nicht wie erwartet ein kleiner Punkt, sondern es passiert nichts! Wird versucht, eines der ASCII-Zeichen 128-159 (&80-&9F) auf die Tastatur zu legen, wird nicht das im Handbuch angegebene ASCII-Zeichen, sondern ein sogenanntes „Erweiterungszeichen", das vorher definiert sein muß, auf die Tastatur gelegt.

15.2 Erweiterungszeichen

Die 32 vorhandenen Erweiterungszeichen (128-159) können wahlfrei mit einem oder mehreren Zeichen (bis 103 Zeichen) belegt werden, wobei die Summe alle Zeichen 104 nicht übersteigen darf. Die ersten 13 Erweiterungszeichen (128-140) sind fest dem 10'er Block zugeordnet (siehe Handbuch Anhang 3/Seite 15). Auf dem Erweiterungszeichen 128 liegt der (ASCII-) Wert &30 (48) und damit die „0", auf 129 das Zeichen „1" (ASCII &31) usw.. Dem Erweiterungszeichen 140 ist ein ganzer Befehl zugeordnet, der bei gleichzeitigem Betätigen der CRTL- und der kleinen ENTER-Taste auf dem Bildschirm ausgegeben und durch das nachgestellte RETURN (&0D = 13) übernommen (ausgeführt) wird (RUN"). Die restlichen Erweiterungszeichen (141-159) sind mit dem ASCII-Code 0 belegt und werden entsprechend ignoriert. Jedes definierte Erweiterungszeichen kann einer beliebigen Taste zugeordnet werden:

```
KEY DEF 71,1,128
```

Das Erweiterungszeichen 128 entspricht dem Zeichen „0" und liegt nun auf der „Z"-Taste.
Da es möglich ist, mehrere Zeichen auf ein Erweiterungszeichen zu legen, bietet es sich an, ganze Befehle oder Befehlsketten auf eine Taste zu legen und Sie dann als Funktionstasten zu benutzen.

15.3 KEY - Belegen von Funktionstasten

Jedes Erweiterungszeichen kann mit

```
KEY erwZchnr, ''zeichenfolge''
```

beliebig definiert werden. Die Nummer des Erweiterungszeichens 'erwZchnr'

muß zwischen 128 und 159 liegen, andernfalls erfolgt die Fehlermeldung „improper argument". Die „zeichenfolge" kann bis zu 103 Zeichen lang sein, allerdings lassen sich dann keine weiteren Erweiterungzeichen definieren, da der vorhandene Speicherplatz damit ausgefüllt ist. Paßt eine Zeichenfolge nicht mehr in den reservierten Speicherplatz, wird ebenfalls ein „improper argument" als Fehlermeldung ausgegeben.

```
KEY 128, ''LIST''
```

Auf die Taste „0" vom 10'er Block (Erweiterungszeichen 128) ist der Befehl „LIST" gelegt worden. Nach Betätigen der Taste „0" vom 10'er Block erscheint nun LIST auf dem Bildschirm und kann mit der Returntaste übernommen und ausgeführt werden. Um den Befehl LIST mit Betätigung der Taste direkt auszuführen, ist dem Befehl noch ein Return (CHR$(13)) nachzustellen.

```
KEY 128, ''LIST''+CHR$(13)
```

Auch Stringvariablen lassen sich in einem KEY-Befehl übergeben:

```
a$=''RUN''+CHR$(13)
KEY 129, a$
```

Die restlichen Erweiterungszeichen 141 bis 159 lassen sich wie beschrieben ebenfalls mit Zeichen oder Zeichenfolgen definieren. Da diese Erweiterungszeichen noch keiner Taste zugeordnet sind, müssen sie einer beliebigen Taste (siehe 15.1) zugeordnet werden.

```
KEY 141,''TEST''
KEY DEF 71,1,&7A,&5A,141
```

Das Erweiterungszeichen 141 ist nun mit „CRTL" und der „Z"-Taste zu erreichen!! Alle Tasten mit Ausnahme der SHIFT- und CRTL-Taste können so als Funktionstasten definiert werden.

15.4 Programmierhilfe

Durch die Möglichkeit, ganze Funktionen auf eine Tasten zu legen, bietet es sich an, BASIC-Schlüsselwörter mit einem Tastendruck abrufbar zu machen. Das nachfolgende Programm belegt 17 Tasten mit BASIC-Befehlen, die zum größten Teil durch gleichzeitiges Betätigen der CRTL- und einer der Zifferntasten ausgegeben werden. Dabei ist vermieden worden, den sehr nützlichen 10'er Block mit Funktionen zu belegen, damit er nicht zum Funktionstastenblock umfunktioniert wird.

Neben der „TAB"-Taste wurde außerdem noch die „CAPS LOCK"-Taste mit 3 Funktionen belegt. Die eigentliche Funktion dieser Taste (Umschalten Groß/Klein) ist damit aufgehoben und somit nicht mehr möglich. Um das Umschalten von Groß- auf Kleinschrift mit der „CAPS LOCK"-Taste in einem Programm zu verhindern, kann die Taste mit

```
KEY DEF 70,0,0,0,0
```

„gesperrt" werden. Ebenso kann die „ESCAPE"-Taste gesperrt werden, so daß ein Programmabbruch nicht mehr möglich ist

```
KEY DEF 66,0,0,0,0
```

```
1000 REM ************************************ M.W.Thoma **
1010 REM *                                               *
1020 REM *                Funktionstasten                *
1030 REM *                                               *
1040 REM *************************************************
1050 KEY DEF 68,0,141,142,143:      'TAB
```

```
1060 KEY DEF 64,1,&31,&21,144:      '1
1070 KEY DEF 65,1,&32,&22,145:      '2
1080 KEY DEF 57,1,&33,&23,146:      '3
1090 KEY DEF 56,1,&34,&24,147:      '4
1100 KEY DEF 49,1,&35,&25,148:      '5
1110 KEY DEF 48,1,&36,&26,149:      '6
1120 KEY DEF 41,1,&37,&27,150:      '7
1130 KEY DEF 40,1,&38,&28,151:      '8
1140 KEY DEF 33,1,&39,&29,152:      '9
1150 KEY DEF 32,1,&30,&5F,153:      '0
1160 KEY DEF 25,1,&2D,&3D,154:      '-
1170 KEY DEF 70,0,155,156,157:      'CAPS LOCK
1180 '
1190 KEY 141,"RUN"+CHR$(13)
1200 KEY 142,"LIST"+CHR$(13)
1210 KEY 143,"MODE 2"+CHR$(13)
1220 KEY 144," DATA"
1230 KEY 145," DRAW"
1240 KEY 146,"=INKEY$"
1250 KEY 147," INPUT"
1260 KEY 148,"=LEFT$("
1270 KEY 149," LOCATE"
1280 KEY 150,"=MID$("
1290 KEY 151," ORIGIN"
1300 KEY 152," PLOT"
1310 KEY 153,"=RIGHT$("
1320 KEY 154," WINDOW#"
1330 KEY 155,"CLS"+CHR$(13)
1340 KEY 156,"AUTO "
1350 KEY 157,"DELETE "
1360 '
1370 MODE 2
1380 PRINT,"Neue Funktionstastenbelegung  aktiv  !!!!!!!!!"
1390 PRINT
1400 PRINT,"Der 10'er  Block  ist unveraendert  geblieben."
1410 PRINT,"Die Funktionen sind mit CRTL  und den normalen"
1420 PRINT,"Zifferntasten erreichbar. Weiterhin liegen auf"
1430 PRINT,"der TABULATOR- und CAPS LOCK-Taste Funktionen."
1440 PRINT
1450 PRINT"     <CRTL>/<1> =  DATA",,
1460 PRINT"     <CRTL>/<2> =  DRAW"
1470 PRINT"     <CRTL>/<3> = =INKEY$",,
1480 PRINT"     <CRTL>/<4> =  INPUT"
1490 PRINT"     <CRTL>/<5> = =LEFT$(",,
1500 PRINT"     <CRTL>/<6> =  LOCATE"
1510 PRINT"     <CRTL>/<7> = =MID$(",,
1520 PRINT"     <CRTL>/<8> =  ORIGIN"
1530 PRINT"     <CRTL>/<9> =  PLOT",,
1540 PRINT"     <CRTL>/<0> = =RIGHT$("
1550 PRINT"     <CRTL>/<-> =  WINDOW#"
1560 PRINT
1570 PRINT"          <TAB> =  RUN+chr$(13)",
1580 PRINT"  <SHIFT>/<TAB> =  LIST+chr$(13)"
1590 PRINT"   <CRTL>/<TAB> =  MODE 2+chr$(13)",
1600 PRINT"         <CAPS> =  CLS+chr$(13)"
```

```
1610 PRINT" <SHIFT>/<CAPS> =  AUTO",,
1620 PRINT"  <CRTL>/<CAPS> =  DELETE"
1630 'NEW
1640 REM ***************************************************
```

15.5 Auslesen der Funktionstastenbelegung

Für die Belegung der Erweiterungszeichen steht der Speicherraum von &B446 bis &B4EF zur Verfügung. Alle Zeichen oder Zeichenfolgen werden hier untergebracht. Mit einem einfachen Programm kann die Belegung erfragt und ausgegeben werden. Innerhalb dieses Speicherbereiches liegen die Erweiterungszeichen immer der Reihe nach geordnet vor (128-159). Nach jeder Definition eines Erweiterungszeichens sortiert das Betriebssystem diesen Bereich neu. Für alle nicht definierten Erweiterungszeichen setzt das Betriebssystem jeweils ein Nullbyte in die Liste. Sind z.B. die Erweiterungszeichen 150 - 157 nicht definiert, stehen zwischen 149 und 158 acht Nullbytes. Mit Hilfe dieser Nullbytes und dem eigentlichen Erweiterungszeichen erkennt der Rechner, um welches Erweiterungszeichen es sich handelt. Mit einem weiteren Byte sind die einzelnen Erweiterungszeichen voneinander getrennt, damit erkannt werden kann, wo das Erweiterungszeichen endet. Das Programm „KEYLIST" zählt die Trenn- und Nullbytes und bestimmt damit das Erweiterungszeichen (Zeile 1110-1120). Alle anderen Zeichen werden normal ausgegeben (Zeile 1140). Trifft das Programm auf eine 13 (Return), gibt es als Information noch „+CHR$(13)" aus (Zeile 1130). Das Programm ist dann beendet, wenn 32 Null- oder Trennbytes gefunden worden sind (Zeile 1110).
Die „Maschinensprache-Programmierer" unter Ihnen sollten das Programm „KEYLIST" in Maschinensprache schreiben und mit in das CPC-BASIC einbinden.

```
1000 REM ************************************ M.W.Thoma **
1010 REM *                                               *
1020 REM *                  .K E Y L I S T               *
1030 REM *                                               *
1040 REM *      zeigt die Belegung der Funktionstasten   *
1050 REM *                                               *
1060 REM ***************************************************
1070 nr=127:i=&B446:ZONE(40):MODE 2
1080 PRINT " > KEYLIST <",,,
1090 x=PEEK(i)
1100 IF x>12 THEN 1130
1110 nr=nr+1:PRINT,:IF nr=160 THEN END
1120 PRINT"<F";nr;"> : ";:GOTO 1150
```

```
1130 IF x=13 THEN PRINT"+chr$(13)";:GOTO 1150
1140 PRINT CHR$(x);
1150 IF i=&B4EF THEN END
1160 i=i+1:GOTO 1090
1170 REM ************************************************
```

16.0 Die Kontroll- und Steuerzeichen

Neben den bisher behandelten Zeichen gibt es noch die „Kontroll- oder Steuerzeichen". Sie belegen die ASCII-Werte 0 bis 31 und erzeugen auf dem Bildschirm normalerweise keine Zeichen, sondern führen eine bestimmte Funktion aus. Alle Steuerzeichen können wie gewöhnliche Zeichen mit

PRINT CHR$(n)

aufgerufen werden. Andererseits liegen alle Steuerzeichen auch auf der Tastatur und sind in Kombination mit der „CRTL"-Taste erreichbar, so daß sie einem String zugeordnet werden können. Fast alle Funktionen, die mit Steuerzeichen ausgeführt werden, sind ohne weiteres mit normalen BASIC-Befehlen realisierbar. In der Regel ist immer ein klarer und eindeutiger BASIC-Befehl einer langen Steuerzeichensequenz vorzuziehen.
Auf den nächsten Seiten finden Sie eine Zusammenstellung aller Kontroll- und Steuerzeichen. Die Liste ist so aufgebaut, daß Sie neben den Steuerzeichen (PRINT CHR$(x);) die Anzahl der zusätzlichen Parameter finden, die notwendig sind, um die Funktion des Steuerzeichens komplett auszuführen. Dann folgt die Bezeichnung des Steuerzeichens (Abkürzung und Klartext) sowie die Tastenkombination, über die das Zeichen erreichbar ist. Die Standardbezeichnung des Steuerzeichens hat teilweise keinen Bezug mehr auf die Funktion. Einige kurze Beispielprogramme sollen die Funktionen verdeutlichen und eine praktische Anwendung zeigen.

PRINT CHR$(0); 0 (NUL) Null z.B. CRTL/1
Hat keine Funktion und wird ignoriert.

PRINT CHR$(1); 1 (SOH) Start Of Heading CRTL/A
Ein vorgestelltes CHR$(1) bewirkt, daß nicht die Funktion eines Steuerzeichens ausgeführt wird, stattdessen erscheint ein grafisches Symbol.

```
10 FOR i = 0 TO 31
20 PRINT CHR$(1);CHR$(i)
30 NEXT i
```

PRINT CHR$(2); 0 (STX) Start Of Text CRTL/B
Schaltet den Textcursor aus.

```
10 PRINT CHR$(2);:INPUT a
```

PRINT CHR$(3); 0 (ETX) End Of Text CRTL/C
Schaltet den Textcursor an.

PRINT CHR$(4); 1 (EOT) End Of Transmission CRTL/D
Bildschirmmodus setzen. Das nachfolgende Zeichen bestimmt den Modus.

PRINT CHR$(4);CHR$(0) = MODE 0
PRINT CHR$(4);CHR$(1) = MODE 1
PRINT CHR$(4);CHR$(2) = MODE 2

PRINT CHR$(5); 1 (ENQ) Enquiry CRTL/E
Gibt nachfolgendes Zeichen auf Grafikcursorposition aus.

```
10 MOVE 320, 200: PRINT CHR$(5); "A"
```

PRINT CHR$(6); 0 (ACK) Acknowledge CRTL/F
Schaltet Bildschirmausgabe wieder an (siehe PRINT CHR$(21);).

PRINT CHR$(7); 0 (BEL) Bell CRTL/G
Gibt einen kurzen Ton aus.

PRINT CHR$(8); 0 (BS) Backspace CRTL/H
Cursor eine Position nach links setzen.

PRINT CHR$(9); 0 (HT) Horizontal Tabulator CRTL/I
Cursor eine Position nach rechts setzen

PRINT CHR$(10); 0 (LF) Line Feed CRTL/J
Cursor eine Position nach unten setzen.

PRINT CHR$(11); 0 (VT) Vertical Tabulator CRTL/K
Cursor eine Position nach oben setzen.

PRINT CHR$(12); 0 (FF) Form Feed CRTL/L
Bildschirm löschen (= CLS).

PRINT CHR$(13); 0 (CR) Carriage Return CR oder CRTL/M
Cursor auf Anfang der aktuellen Zeile setzen.

PRINT CHR$(14); 1 (SO) Shift Out CRTL/N

Setzt die PAPER-Farbe auf den nachfolgenden Parameter (= PAPER).

```
10 PRINT CHR$(14);CHR$(1)
```

PRINT CHR$(15); 1 (SI) Shift In CRTL/O

Setzt die PEN-Farbe auf den nachfolgenden Parameter (= PEN).

```
10 PRINT CHR$(15);CHR$(2)
```

PRINT CHR$(16); 0 (DLE) Data Link Escape CRTL/P

Zeichen auf Cursorposition löschen.

```
10 FOR i = 1 TO 70:PRINT "*";:NEXT i
20 FOR j = 1 TO 70
30 PRINT CHR$(16);:REM löscht Zeichen
40 PRINT CHR$(8); :REM Cursor n. links
50 NEXT j
60 PRINT CHR$(7); :REM kurzer Ton
70 GOTO 10
```

PRINT CHR$(17); 0 (DC1) Divice Control 1 CRTL/Q

Löscht aktuelle Zeile vom Anfang bis einschließlich Cursorposition.

PRINT CHR$(18); 0 (DC2) Divice Control 2 CRTL/R

Löscht aktuelle Zeile von der Cursorposition bis zum Ende.

```
10 mode 2
20 FOR i = 1 TO 70:PRINT "*";:NEXT i
30 LOCATE 1, 1
40 PRINT CHR$(18)
```

PRINT CHR$(19); 0 (DC3) Divice Control 3 CRTL/S

Löscht Bildschirmbereich von der ersten Zeile bis zur Cursorposition.

PRINT CHR$(20); 0 (DC4) Divice Control 4 CRTL/T
Löscht Bildschirmbereich von der Cursorposition bis zur letzten Zeile.

```
10 MODE 2
20 FOR i = 1 to 20:PRINT "Hallo":next i
30 locate 1, 10
40 PRINT CHR$(20)
```

PRINT CHR$(21); 0 (NAK) Negative Acknowledge CRTL/U
Schaltet Bildschirmausgabe solange aus, bis ein CHR$(6) gesendet wird.

```
10 MODE 2
20 PRINT CHR$(21);
30 PRINT "unsichtbar";
40 PRINT CHR$(6);
50 PRINT "sichtbar";
60 GOTO 20
```

```
10 PRINT "Codewort :";
20 PRINT CHR$(21);
30 INPUT kw$
40 PRINT CHR$(6);
50 PRINT "Codewort war ";kw$
```

PRINT CHR$(22); 1 (SYN) Synchronous Idle CRTL/V
Transparent-Modus (siehe 2.6)
PRINT CHR$(22);CHR$(0) = Transp.-Modus aus
PRINT CHR$(22);CHR$(1) = Transp.-Modus an

PRINT CHR$(23); 1 (ETB) End of Transm. Block CRTL/W
Grafikfarbstiftmodus setzen.

PRINT CHR$(24); 0 (CAN) Cancel CRTL/X
Schaltet Inverse-Modus ein oder aus (siehe 4.4)

PRINT CHR$(25); 9 (EM) End Of Medium CRTL/Y
Definieren eines neuen Zeichens (= SYMBOL). Es folgen 9 Daten, die das Zeichen und die Zeichenform bestimmen (siehe 4.4).

```
10 SYMBOL AFTER 32
20 PRINT CHR$(25);CHR$(94);CHR$(28);
30 PRINT CHR$(35);CHR$(99);CHR$(108);
40 PRINT CHR$(99);CHR$(99);CHR$(110);
50 PRINT CHR$(96)
60 FOR i = 1 TO 100
70 PRINT CHR$(94);
80 NEXT i
```

PRINT CHR$(26); 4 (SUB) Substitute CRTL/Z

Definiert das Fenster 0 (WINDOW #0). Die 4 folgenden Parameter bestimmen die Größe des Fensters (siehe 5.0).

```
10 PRINT CHR$(26);CHR$(5);CHR$(15);
20 PRINT CHR$(5);CHR$(15)
```

PRINT CHR$(27); 0 (ESC) Escape CRTL/(

Hat keinen Einfluß und wird ignoriert.

PRINT CHR$(28); 2 (FS) File Separator CRTL//

Füllt Farbfaß mit Farben und entspricht daher dem Befehl INK (siehe 2.3).

PRINT CHR$(29); 2 (GS) Group Separator CRTL/)

Bestimmt die Farbe(n) des Borders. Gleichwertig mit BORDER (siehe 2.4).

PRINT CHR$(29);CHR$(26);CHR$(26)

PRINT CHR$(30); 0 (RS) Record Separator CRTL/^

Setzt Cursor in die obere, linke Ecke des Bildschirmbereichs. Entspricht LOCATE 1,1.

PRINT CHR$(31); 2 (US) Unit Separator CRTL/0

Setzt den Cursor auf die angegebene Position. Die beiden Parameter bestimmen Spalte und Zeile. Gleichwertig mit dem LOCATE-Befehl.

PRINT CHR$(31);CHR$(20);CHR$(10);''Test''

17.0 Fehlerbehandlung

Immer wenn das Programm auf eine Programmzeile trifft, die der BASIC-Interpreter nicht übersetzen kann oder wenn ein logischer Fehler (z.B. es fehlen Daten) vorliegt, beendet der Interpreter seine Arbeit und gibt eine der 31 möglichen Fehlermeldungen aus. Jeder Fehlermeldung ist eine Nummer von 1 bis 30 zugeordnet, die Sie der folgenden Tabelle mit entsprechenden Klartext entnehmen können.

0	Unknown error	Unbekannter Fehler
1	Unexpected Next	Unerwartetes NEXT
2	Syntax error	Syntaxfehler
3	Unexpected RETURN	Unerwartetes RETURN
4	DATA exhausted	Daten fehlen
5	Improper argument	Unzulässiges Argument
6	Overflow	Über-/Unterlauf
7	Memory full	Speicherende erreicht
8	Line does not exist	Zeile nicht vorhanden
9	Subscript out of range	Index außerh. d. Bereiches
10	Array already dimensioned	Array schon definiert
11	Division by zero	Division durch Null ''0''
12	Invalid direct command	Falsches Direktkommando
13	Type mismatch	Falscher Datentyp
14	String space full	Stringbereich überfüllt
15	String too lang	String zu lang
16	String expression too complex	String zu komplex
17	Cannot CONTinue	CONT nicht möglich
18	Unknown user function	Unbekannte Funktion
19	RESUME missing	RESUME fehlt
20	Unexpected RESUME	Unerwartetes RESUME
21	Direct command found	Direktkommando gefunden
22	Operand missing	Operand fehlt
23	Line too long	Programmzeile zu lang
24	EOF met	Dateiende angetroffen
25	File type error	Falscher Dateityp
26	NEXT missing	NEXT fehlt
27	File already open	Datei ist schon offen
28	Unknown command	Unbekanntes Kommando
29	WEND missing	WEND fehlt
30	Unexpected WEND	Unerwartetes WEND

Alle diese Fehlermeldungen lassen sich mit dem Befehl

ERROR fehlernr

aufrufen. Die „fehlernr" muß dabei einen Wert zwischen 0 und 30 besitzen. Durch die Eingabe von z.B.

ERROR 11

gibt der Rechner die Meldung „Division by zero" aus und bricht seine Arbeit mit einer „Ready"-Meldung ab.

Der CPC-464 bietet einige komfortable Möglichkeiten, um einen auftretenden Fehler zu bestimmen (welcher Fehler ?) und zu lokalisieren (wo aufgetreten ?). Es ist sogar möglich, trotz eines aufgetretenen Fehlers das Programm fortzuführen. Nach der Programmzeile

1000 ON ERROR GOTO 2000

springt das Programm bei einem auftretenden Fehler automatisch zur angegebenen Zeilennummer (2000). Ab dieser Zeilennummer (2000) muß dann ein Programmteil stehen, in dem die Fehlerbehandlung folgt. Als Entscheidungskriterien stehen zwei Systemvariablen zur Verfügung, die Auskunft geben, in welcher Zeile welcher Fehler aufgetreten ist. Dabei stehen in den (System-) Variablen

ERR	die Fehlernummer (1-30) und in
ERL	die Zeilennummer, wo der Fehler aufgetreten ist.

Hierzu ein kleines Beispiel:

```
1000 ON ERROR GOTO 2000
1010 a="String"
1020 PRINT "Programmende"
1030 END
2000 PRINT "Fehlernummer";ERR;"in Zeile";ERL
2010 END
```

Das Programm trifft in Zeile 1010 auf einen Zuweisungsfehler, springt automatisch in die Programmzeile 2000 und gibt die Meldung

Fehlernummer 13 in Zeile 1010

aus. Das Programm endet mit dem END-Befehl in Zeile 2010. Statt das Programm mit END in Zeile 2010 zu beenden, könnte trotz des aufgetretenen Fehlers das Programm mit dem nächsten Befehl, der dem fehlerhaften Befehl folgt, fortgesetzt werden. Durch den Befehl

RESUME NEXT

(resume = wieder aufnehmen, next = nächste) wird die Fehlerbehandlungsroutine (ab Zeile 2000) beendet und gleichzeitig angewiesen, das Programm mit dem nächsten Befehl fortzuführen. Ersetzen Sie die Programmzeile 2010 mit

```
2010 RESUME NEXT
```

Nach der Bearbeitung des Fehlers wird das Programm dann in der Programmzeile 1020 fortgesetzt und endet mit dem END in Zeile 1030. Selbstverständlich ist damit nicht der eigentliche Fehler in der Programmzeile 1010 behoben, aber diese Art der Fehlerbehandlung schützt vor unliebsamen Programmabbrüchen und damit vor Datenverlust.

Die zweite Möglichkeit, ein Programm trotz aufgetretenem Fehler fortzusetzen, bietet der Befehl

RESUME zeilennr

Mit RESUME 'zeilennr' kann das Programm an einer beliebigen Stelle fortgesetzt werden. So könnten z.B. fehlerhafte Eingaben wiederholt werden, ohne daß ein Programmabbruch stattfindet. Auch hierzu ein kleines Beispiel:

```
1000 ON ERROR GOTO 2000
1010 PRINT "Divisionsprogramm"
1020 INPUT "Zaehler :";z
1030 INPUT "Nenner  :";n
1040 PRINT "Ergebnis";z/n
1050 GOTO 1020
1060"
2000 PRINT "*** Fehler bei der Eingabe !! ***"
2010 RESUME 1020
```

Treten Fehler auf, so können sie (bei diesem fehlerfreien Programm) nur bei der Eingabe des Zählers und des Nenners entstehen (z.B. Nenner = 0, Division by zero etc). Die Fehlerbehandlungsroutine gibt eine Meldung aus und springt nach Zeile 1020, wo die Werte neu eingegeben werden müssen.

18.0 Interrupt - Unterbrechungen

Etwas ganz Besonderes bietet der CPC-464 mit seiner überaus einfachen Interruptbedienung. Interruptgesteuerte Programme waren in der Vergangenheit immer nur Sachen für absolute Maschinensprachespezialisten. Die Programmierung war sehr umständlich (teilweise kaum möglich) und extrem kompliziert. Beim CPC-464 ist dieses anders: Einfache, klare Befehlsstrukturen und das mit BASIC-Befehlen.
Interrupt heißt nichts anderes als Unterbrechung. Doch wozu Unterbrechungen ? Stellen Sie sich bitte einmal folgendes vor: In einem BASIC-Programm hat sich eine unendliche Schleife eingeschlichen:

```
10 GOTO 10
```

Das Programm würde nun bis zum jüngsten Tag in dieser Zeile „kreisen", wenn nicht die Möglichkeit bestünde, das Programm mit der Escape-Taste zu stoppen. Doch woher weiß der Rechner, daß die Escape-Taste gedrückt ist? Er unterbricht das laufende Programm (10 GOTO 10) in bestimmten Zeitabständen und prüft, ob die Escape-Taste gedrückt ist. Eine solche Unterbrechung (Interrupt) findet 50 mal in der Sekunde statt und es wird dann ein bestimmtes Prüfprogramm (Interruptroutine) ausgeführt. Nachdem die Interruptroutine durchgeführt ist (z.B. Tastatur prüfen), wird das Programm an der Stelle fortgesetzt, wo es unterbrochen wurde oder aber abgebrochen, wenn die Escape-Taste betätigt war. Allgemein kann gesagt werden, daß ein Interrupt eine Unterbrechung des laufenden Programms zu einer exakt bestimmten Zeit ist, die genutzt werden kann, um eine andere, vom eigentlichen Programm abweichende Routine durchzuführen.

Damit ein Interrupt zu einer exakt bestimmten Zeit eingeleitet werden kann, benötigt der Rechner eine Uhr (Timer), die sehr genau laufen muß. Der Zustand dieser zentralen Uhr läßt sich mit dem Befehl

```
TIME
```

abfragen. Der wiedergegebene numerische Wert zeigt die seit dem Einschalten (Reset) vergangene Zeit in 1/300 Sekunden an.

```
PRINT TIME
17567
```

Mit einfachen Umrechnungen läßt sich die wiedergegebene Zeit in verschiedene Zeiteinheiten formen.

PRINT TIME	Zeit in 1/300 Sekunden
PRINT TIME/3	Zeit in 1/100 Sekunden
PRINT TIME/30	Zeit in 1/10 Sekunden
PRINT TIME/300	Zeit in Sekunden
PRINT TIME/18000	Zeit in Minuten
PRINT TIME/1080000	Zeit in Stunden

Mit Hilfe der zentralen Uhr können sehr genaue (1/300 Sekunde) Zeitdifferenzen gemessen werden. An einem ganz kleinen Reaktionstest möchte ich Ihnen das Prinzip erklären.

```
10 PRINT "Bitte eine Taste druecken"
20 t=TIME
30 in$=INKEY$:IF in$="" THEN 30
40 PRINT INT((TIME-t)/3);"/100 Sekunden"
```

In dem Beispiel wird die Zeit gemessen, die von der Aufforderung, eine Taste zu betätigen bis zur Betätigung vergangen ist. In Zeile 20 wird die aktuelle Zeit in der Variablen „t" gespeichert. In der Zeile 30 wartet der Rechner auf die Betätigung einer Taste. Ist eine Taste betätigt, wird die aktuelle Zeit mit der gemerkten Zeit ('t') verglichen. Die Differenz gibt die vergangene Zeit in 1/300 Sekunden an. Diese Zeit wird in 1/100 Sekunden umgerechnet und ausgegeben.

Anzumerken ist noch, daß die zentrale Uhr bei einer Kassetten- oder Diskettenoperation (LOAD, SAVE, etc.) stoppt. Dieses ist bedauerlich, da so die zentrale Uhr nicht für eine kontinuierliche Anzeige der Tageszeit geeignet ist.

Vier weitere Timer, die von einem bestimmten Wert abwärts gegen Null zählen, sind im CPC-464 vorhanden. Diese Timer sind die eigentlichen „Interrupt-Timer". Immer dann, wenn einer dieser Timer bei Null angelangt ist, wird in eine vorher bestimmte Interruptroutine gesprungen.

18.1 AFTER - einmaliger Aufruf

Die erste im CPC-464 vorhandene Interruptmöglichkeit bildet der AFTER-Befehl (after = nach). Nach einer bestimmten Zeit wird einmalig in eine Inter-

ruptroutine gesprungen. Die Interruptroutine ist ein normales BASIC-Unterprogramm, wie es vom GOSUB-Befehl her bekannt ist.

AFTER zeit,timer GOSUB zeilenr

Nach dem eigentlichen Befehl AFTER folgt die Zeit in 1/50 Sekunden, die vergehen soll, bis die Interruptroutine aufgerufen wird. Da insgesamt 4 verschiedene Timer zur Verfügung stehen, wählt „timer" einen dieser Timer aus. Die Timer sind von 0 bis 3 durchnummeriert. Mit „GOSUB zeilennr" ist der Beginn der eigentlichen Interruptroutine bestimmt. Hierzu ein konkretes Beispiel:

```
10 AFTER 500,0 GOSUB 50
20 PRINT "Hauptproamm laueft"
30 IF x=0 THEN 20
40 end
50 "Interruptroutine
60 PRINT CHR$(24);"Interruptroutine laeuft";CHR$(24)
70 x=1:RETURN
```

Nach 10 Sekunden (500/50 = 10) soll das Unterprogramm ab Zeile 50 aufgerufen werden. Als Zeitgeber dient der Timer 0. Solange diese 10 Sekunden noch nicht vergangen sind, arbeitet das Programm normal weiter und „kreist" in den Programmzeilen 20 und 30. Hat der Timer auf Null heruntergezählt, springt das Programm automatisch in das Unterprogramm ab Zeile 50. Dieses wird abgearbeitet (Textausgabe und x = 1 setzen), bis ein RETURN gefunden wird. Durch diesen RETURN-Befehl springt das Programm automatisch wieder an die Stelle, wo es vorher unterbrochen wurde. Da „x" nach Aufruf des Unterprogramms den Wert 1 besitzt, endet das Programm in Zeile 40.

Da insgesamt 4 voneinander unabhängige Timer vorhanden sind, besteht die Möglichkeit, 4 verschiedene Interruptzeiten festzulegen.

```
10 AFTER 100,0 GOSUB 1000
20 AFTER 200,1 GOSUB 2000
30 AFTER 300,2 GOSUB 3000
40 AFTER 400,3 GOSUB 1000
50 .......
```

Verschiedene AFTER-Befehle können durchaus das gleiche Unterprogramm nutzen (Zeile 40). Treten zwei Interrupts zur selben Zeit auf,

```
10 AFTER 100,0 GOSUB 1000
20 AFTER 100,1 GOSUB 2000
30 .......
```

wird zuerst die Interruptroutine mit der höheren Timernummer ausgeführt. Die größte Timernummer besitzt auch die größte Priorität:

Timer 3	höchste Priorität
Timer 2	
Timer 1	
Timer 0	niedrigste Priorität

18.2 EVERY - ständiger Aufruf

Im Gegensatz zum AFTER-Befehl (einmaliger Aufruf) wird ein mit EVERY angesprochenes Interrupt-Unterprogramm ständig nach einer bestimmten Zeit aufgerufen (ever = immer, stets). Nach Ausführung des Unterprogramms setzt sich der Timer automatisch wieder auf den vorher bestimmten Wert und zählt erneut gegen Null. Ist er bei Null angelangt, springt das Programm wieder in das Interrupt-Unterprogramm.

EVERY zeit,timer GOSUB zeilnr

Der Aufbau des Befehls ist vollständig identisch mit dem des AFTER-Befehls.

```
10 EVERY 50,0 GOSUB 60
20 FOR i = 1 TO 5000
30 PRINT "Haupt", i,
40 NEXT i
50 END
60 'Interruptroutine
70 PRINT:PRINT "Inter"
80 RETURN
```

Obwohl das Programm in der FOR-TO-NEXT-Schleife kreist, wird es regelmäßig nach jeder Sekunde (50/50 = 1) durch die Interruptroutine unterbrochen.
Ein etwas umfangreicheres Beispiel zeigt eine der Anwendungsmöglichkeiten der Interrupt-Befehle. Eine Uhr, in diesem Fall eine Analog-Uhr (Zeigeruhr),

wird durch einen Interrupt pro Sekunde gesteuert. Da die Interruptroutine sehr exakt aufgerufen wird, läuft die Uhr mit einer hohen Genauigkeit. Das eigentliche Interrupt-Unterprogramm befindet sich in den Programmzeilen 1590-1850. Hier wird jede Sekunde der Sekundenpunkt neu gesetzt. Ist eine volle Minute erreicht, wird der Minutenzeiger um eine Minute vorgestellt. Der Stundenzeiger folgt alle 12 Minuten um eine Zeiteinheit. Damit ist ein wesentlich besseres Erscheinungsbild gewährleistet. Nach Eingabe der Uhrzeit und dem Aufbau der Uhr folgt in Programmzeile 1450 die Aktivierung des Interrupt-Timers 0. Dann kreist das Programm in der Zeile 1520 und wird regelmäßig durch den Interrupt unterbrochen.

```
1000 REM ************************************* M.W.Thoma **
1010 REM *                                                *
1020 REM *          Analog-Uhr fuer den CPC 464           *
1030 REM *                                                *
1040 REM ***************************************************
1050 MODE 1
1060 PRINT "Analog-Uhr fuer den Schneider CPC - 464"
1070 LOCATE 4,4:INPUT"Stunde (0-11) :";t1
1080 IF t1<0 OR t1>11 THEN 1070
1090 LOCATE 4,5:INPUT"Minute (0-59) :";t2
1100 IF t2<0 OR t2>59 THEN 1090
1110 m=t2-1:m1=1
1120 h=t1*5+INT(t2/12)
1130 MODE 2
1140 LOCATE 13,1
1150 PRINT "A N A L O G  -  U H R   ";
1160 PRINT "F U E R   D E N   C P C - 4 6 4"
1170 LOCATE 17,24
1180 PRINT CHR$(164);" Manfred Walter Thoma    ";
1190 PRINT "Hamburg   (BRD 1985)"
1200 '
1210 REM ***************************************************
1220 REM *                                                *
1230 REM *           Zeichnen der Uhr (Ziffern)           *
1240 REM *                                                *
1250 REM ***************************************************
1260 ORIGIN 320,200
1270 f=1:xm=0:ym=0:xr=150:yr=150:wa=0:we=360:GOSUB 60000
1280 FOR i= 0 TO 360 STEP 6
1290 MOVE 150*COS(i),150*SIN(i)
1300 IF i/30=INT(i/30) THEN DRAW 130*COS(i),130*SIN(i)
1310 DRAW 140*COS(i),140*SIN(i)
1320 NEXT i
1330 LOCATE 40,6 :PRINT"12"
1340 LOCATE 32,7 :PRINT"11"
1350 LOCATE 48,7 :PRINT"1"
1360 LOCATE 28,10:PRINT"10"
1370 LOCATE 53,10:PRINT"2"
1380 LOCATE 26,13:PRINT"9"
```

```
1390 LOCATE 55,13:PRINT"3"
1400 LOCATE 28,16:PRINT"8"
1410 LOCATE 53,16:PRINT"4"
1420 LOCATE 33,19:PRINT"7"
1430 LOCATE 48,19:PRINT"5"
1440 LOCATE 41,20:PRINT"6"
1450 EVERY 50,0  GOSUB 1540
1460 '
1470 REM ***************************************************
1480 REM *                                                 *
1490 REM *          Endlosschleife !!!!!!!!!!!!!!!         *
1500 REM *                                                 *
1510 REM ***************************************************
1520 GOTO 1520
1530 '
1540 REM ***************************************************
1550 REM *                                                 *
1560 REM *         Interrupt-Routine Zeiger setzen         *
1570 REM *                                                 *
1580 REM ***************************************************
1590 MOVE 152*COS(90-s*6),152*SIN(90-s*6)
1600 DRAW 160*COS(90-s*6),160*SIN(90-s*6),0
1610 s=s+1:IF s=60 THEN s=0:m1=1
1620 MOVE 152*COS(90-s*6),152*SIN(90-s*6)
1630 DRAW 160*COS(90-s*6),160*SIN(90-s*6),1
1640 IF m1=0 THEN RETURN
1650 m1=0:MOVE 0,0
1660 DRAW 70*COS(94-m*6),70*SIN(94-m*6),0
1670 DRAW 100*COS(90-m*6),100*SIN(90-m*6),0
1680 DRAW 70*COS(85-m*6),70*SIN(85-m*6),0
1690 DRAW 0,0,0
1700 m=m+1:IF M=60 THEN m=0:h1=1
1710 MOVE 0,0:DRAW 70*COS(94-m*6),70*SIN(94-m*6),1
1720 DRAW 100*COS(90-m*6),100*SIN(90-m*6),1
1730 DRAW 70*COS(85-m*6),70*SIN(85-m*6),1
1740 DRAW 0,0,1
1750 IF h1=0 AND m/12 <> INT(m/12) THEN 1810
1760 DRAW 35*COS(80-h*6),35*SIN(80-h*6),0
1770 DRAW 70*COS(90-h*6),70*SIN(90-h*6),0
1780 DRAW 35*COS(100-h*6),35*SIN(100-h*6),0
1790 DRAW 0,0,0
1800 h=h+1:h1=0:IF h=60 THEN h=0
1810 DRAW 35*COS(80-h*6),35*SIN(80-h*6),1
1820 DRAW 70*COS(90-h*6),70*SIN(90-h*6),1
1830 DRAW 35*COS(100-h*6),35*SIN(100-h*6),1
1840 DRAW 0,0,1
1850 RETURN
1860 '
60000 REM ***************************************************
60010 REM *          Ellipse- und Ellipsenteile             *
60020 REM *                                                 *
60030 REM *      wa = Winkelanfang ( < we )                 *
60040 REM *      we = Winkelende   ( > wa )                 *
60050 REM *      xm = X-Koordinate des Mittelpunktes        *
60060 REM *      ym = Y-Koordinate des Mittelpunktes        *
```

```
60070 REM *       xr = X-Radius                                *
60080 REM *       yr = Y-Radius                                *
60090 REM *        f = Zeichenfarbe                            *
60100 REM *                                                    *
60110 REM ***************************************************
60120 DEG
60130 FOR winkel = wa TO we
60140 x=xm+xr*COS(winkel)
60150 y=ym+yr*SIN(winkel)
60160 PLOT x,y,f
60170 NEXT winkel
60180 RETURN
60190 REM ***************************************************
```

Selbstverständlich können wieder bis zu 4 verschiedene Interrupts mit entsprechenden Unterprogrammen erstellt werden. Doch es tauchen nicht unerhebliche Probleme auf, wenn die Interruptroutine mehr Zeit benötigt, als die vorher bestimmte Interruptzeit. Zum Beispiel: Ein Interrupt-Unterprogramm soll alle 5 Sekunden aufgerufen und abgearbeitet werden, doch die Bearbeitungszeit für das gesamte Unterprogramm beträgt schon 6 Sekunden. Das Interrupt-Unterprogramm wird trotz der längeren Zeit komplett bearbeitet, aber das Programm springt dann sofort wieder ins Unterprogramm, da es ja schon wieder aufgerufen worden ist. Probieren Sie bitte folgendes Beispiel aus:

```
10 EVERY 50, 0 GOSUB 50
20 PRINT CHR$(24);"Hauptprg.";CHR$(24);
30 GOTO 20
40 'Interruptroutine
50 FOR i = 1 TO 100
60 PRINT"Interprg.";
70 NEXT i
80 PRINT "INTER-ENDE"
90 RETURN
```

Das Hauptprogramm (Zeile 20-30) arbeitet insgesamt nur die erste Sekunde, dann hängt sich das Programm in der Interruptroutine (ab Zeile 50) auf, da dieses Unterprogramm mehr als 1 Sekunde benötigt! Beachten Sie diese Tatsache bitte bei Ihren Interrupt-Unterprogrammen.

Ein ähnliches Problem tritt beim Arbeiten mit mehreren, verschachtelten Interrupt-Unterprogrammen auf. Auch hier zunächst ein kleines Beispiel:

```
10 EVERY 10, 0 GOSUB 60
20 EVERY 20, 1 GOSUB 90
```

```
30 PRINT "Hauptprg."
40 GOTO 30
50 'Interrupt Timer 0
60 Print"Inter 1"
70 RETURN
80 'Interrupt Timer 1
90 FOR i=1 TO 100:PRINT"Inter 2 ";:NEXT i
100 PRINT "ENDE INTER 2"
110 RETURN
```

Auch in diesem Fall hängt sich das Programm in der Interruptroutine des Timers 1 (ab Zeile 90) auf. An dieser Stelle kommen die unterschiedlichen Prioritäten der einzelnen Timer voll zum Tragen (siehe 18.1). Wie schon erwähnt, steigt die Priorität mit der Timernummer. Obwohl während der Interruptroutine des Timers 1 (ab Zeile 90) ein Interrupt des Timers 0 eintritt, springt das Programm nicht in die Interruptroutine des Timers 0 (ab Zeile 60), da der laufende Interrupt (Timer 1) eine höhere Priorität besitzt! Ändern Sie allerdings die Interrupt-Befehle in

```
10 EVERY 10, 1 GOSUB 60
20 EVERY 20, 0 GOSUB 90
```

so besitzt nun die Interruptroutine ab Zeile 60 die höhere Priorität. Probieren Sie das geänderte Programm bitte aus.
Ganz deutlich ist zu erkennen, daß in diesem Fall die Interruptroutine ab Zeile 60 die Interruptroutine ab Zeile 90 regelmäßig unterbricht. Nach der Unterbrechung setzt das Programm seine Arbeit in dem unterbrochenen Unterprogramm fort!
Allgemein läßt sich sagen, daß eine (zeitlich) lange Interruptroutine eine niedrigere Priorität besitzen sollte. Planen Sie Ihre Interrupt-Unterprogramme stets sehr sorgsam, damit Sie keine unerwünschten Ergebnisse erhalten.

18.3 REMAIN - Timer abschalten

Ein mit AFTER oder EVERY aktivierter Interrupt läßt sich zu jeder Zeit abschalten (deaktivieren). Abschalten bedeutet, daß ab sofort nicht mehr in das entsprechende Unterprogramm gesprungen wird. Ein abgeschalteter Interrupt-Timer bleibt dann solange inaktiv, bis er mit einem AFTER- oder EVERY-Befehl neu eingeschaltet wird. Der Befehl

REMAIN (timer)

schaltet den mit „timer" (0-3) bezeichneten Zeitgeber aus. „REMAIN (timer)" liefert einen numerischen Wert, der die verbliebene Zeit bis zum nächsten Interrupt wiedergibt (remain = Rest). Ist der zurückgegebenen Wert '0',

```
PRINT REMAIN (0)
 0
```

war der angesprochene Timer nicht aktiviert. Da die REMAIN-Funktion einen numerischen Wert zurückgibt, muß ein korrekter Abschaltbefehl folgendermaßen aussehen (hier für Timer 2):

```
PRINT REMAIN (2)      oder
variable=REMAIN (2)
```

Der Vorteil, einen Timer mit einer Zuweisung an eine numerische Variable abzuschalten, liegt darin, daß keine störende Ausgabe auf dem Bildschirm erfolgt. Die verbliebene Zeit steht dann in der Variablen.
Ein weiteres Interrupt-Demoprogramm faßt das Behandelte noch einmal zusammen. Neben dem Hauptprogramm (Ausgabe von Zufallszeichen) laufen drei weitere Programme „gleichzeitig". Jedem eigenständigen Programm ist ein unterschiedliches Fenster zugeordnet, so daß die Arbeitsweise genau betrachtet werden kann. Die Interrupt-Unterprogramme bestehen aus zwei Routinen zur Sinus- und Cosinusberechnung sowie einem Sekundenzähler. Mit dem Erreichen von 360 Grad stellen die Unterprogramme zur Sinus- und Cosinusberechnung ihre Arbeit ein und schalten sich selbständig ab (Zeile 1240 und 1300). Nach 3 Minuten (240 Sekunden) setzt der Sekundenzähler die Variable 'e' auf '1' und das Hauptprogramm beendet das Programm in Zeile 1190.

```
1000 REM ***************************************************
1010 REM *                                                 *
1020 REM *                Interrupt-Demo                   *
1030 REM *                                                 *
1040 REM ***************************************************
1050 MODE 2
1060 WINDOW#0,1,80,1,1
1070 WINDOW#1,1,40,2,20
1080 WINDOW#2,41,80,2,20
1090 WINDOW#3,1,80,21,25
1100 PEN 0:PAPER 1:CLS
1110 PEN#3,0:PAPER#3,1:CLS#3
1120 PRINT"Time :";TAB(40);"Hauptprogramm laeuft unten"
1130 PRINT#1,"Interruptroutine 1  <Sinus>"
1140 PRINT#2,"Interruptroutine 2  <Cosinus>"
1150 EVERY 30,0 GOSUB 1210
```

```
1160 EVERY 15,1 GOSUB 1270
1170 EVERY 50,2 GOSUB 1330
1180 IF e=0 THEN PRINT#3,CHR$(INT(RND(1)*96)+32);:GOTO 1180
1190 END
1200 '
1210 'Unterprogramm Timer 0
1220 s=s+1
1230 PRINT#1,USING "####.####";s,SIN(s)
1240 IF s=360 THEN PRINT#1,"*** ENDE ***":x=REMAIN(0)
1250 RETURN
1260 '
1270 'Unterprogramm Timer 1
1280 c=c+1
1290 PRINT#2,USING "####.####";c,COS(c)
1300 IF c=360 THEN PRINT#2,"*** ENDE ***":x=REMAIN(1)
1310 RETURN
1320 '
1330 'Unterprogramm Timer 2
1340 t=t+1
1350 LOCATE 7,1:PRINT USING "####";t
1360 IF t=240 THEN e=1:REM Ende markieren !!!!
1370 RETURN
1380 REM ************************************************
```

18.4 DI und EI - zeitweiliges Ausschalten

Zwei Befehle (die auch im Befehlssatz des Z80-Mikroprozessors vorhanden sind) ermöglichen es, alle Interrupts für eine gewisse Zeit zu verhindern.

DI sperre Interrupt und
EI ermögliche Interrupt

Die Anwendung des DI- und EI-Befehls ist immer dann notwendig, wenn ein bestimmter Programmteil ohne Unterbrechung durch einen laufenden Interrupt abgearbeitet werden soll. Nach einem DI-Befehl bleiben alle Interrupts (außer BREAK mit der Escape-Taste) gesperrt, bis ein EI-Befehl folgt.

```
10 EVERY 10, 0 GOSUB 90
10 PRINT "Interrupt möglich"
30 FOR i=1 TO 5000:NEXT i
40 PRINT "Interrupt gesperrt"
50 DI
60 FOR i=1 TO 5000:NEXT i
70 EI
80 GOTO 20
90 t=t+1:PRINT t;:RETURN
```

Es ist sofort ersichtlich, daß nach dem DI-Befehl in Zeile 50 keine weitere Unterbrechung stattfindet. Allerdings tritt nach dem EI-Befehl in Zeile 70 etwas merkwürdiges auf: Es werden stattgefundene, aber nicht ausgeführte Interruptaufrufe (bedingt durch DI) nachgeholt!! Das heißt, der Rechner merkt sich alle Aufrufe von Interruptroutinen und holt sie gegebenenfalls nach.

Insgesamt bietet die Interrupttechnik eine Vielzahl von Anwendungsmöglichkeiten. Nach anfänglichen Schwierigkeiten ist der Umgang mit den Interrupt-Befehlen recht einfach. Die Interrupttechnik sollte jedoch nur dann benutzt werden, wenn die Anwendung wirklich notwendig ist, da die Übersicht des Programmablaufs doch erheblich beeinträchtigt wird.

19.0 Externe Datenspeicher

An dieser Stelle folgt eine kurze Einführung in die Handhabung der Kassette und Diskette. Leider ist es in diesem ersten Band (bedingt durch den Umfang) nicht möglich, alle Probleme der Datenverwaltung ausführlich zu behandeln. Ich verweise deshalb auf den zweiten Band der Reihe „CPC-464 Praxis", in dem sehr umfangreich auf diesen Bereich eingegegangen wird. Die folgende, kurze Einführung beschränkt sich auf die in diesem Band benutzten Kassetten- oder Diskettenoperationen.

Nach Abschalten der Betriebsspannung verlieren alle RAM-Bausteine ihren Inhalt (siehe 1.0), so daß Programme und Daten, die sich in diesen Bausteinen befinden, verloren gehen. Mit Hilfe von externen Datenspeichern läßt sich der Inhalt von RAM-Bausteinen sichern (SAVE = sichern). Zu jeder beliebigen Zeit können diese Daten (Programme, Daten) wieder in die RAM-Bausteine geladen werden (LOAD = laden). In der Konsole des CPC-464 befindet sich ein solcher Datenträger fest eingebaut: Der Datacorder (Kassettenrecorder). Auf handelsüblichen Musikkassetten können beliebige Daten gespeichert und von ihm wieder in den Computer geladen werden. Die Kombination Computer, Monitor und Datacorder bildet einen kompletten, funktionsfähigen (Home-) Computer, d.h. Sie müssen keine weiteren Zusatzgeräte kaufen. In dieser Kombination ähnelt der CPC-464 stark dem legendären PET (Personal Electronic Transaction computer, Warenzeichen der Firma COMMODORE) der siebziger Jahre.

In der Konsole des CPC-664 ist im Gegensatz zum CPC-464 fest die Diskettenstation DDI-1 eingebaut. Alle Angaben zur Kassette beziehen sich deshalb nur auf den CPC-464.

Durch die immer weiter sinkenden Preise für Computer und Computerperipherie (mit dem Computer verbundene Geräte, die von ihm kontrolliert werden) hat sich in der Vergangenheit die Diskettenstation auch im Homecomputerbereich durchgesetzt. Eine Diskettenstation zeichnet sich durch sehr zuverlässiges und vor allem sehr schnelles Arbeiten aus.

19.1 Kassette und Diskette

Das Band der Kassette ist mit einer magnetisierbaren Schicht belegt, deren Teilchen sich unterschiedlich magnetisieren lassen. Je nach Art der Magnetisierung kann festgestellt werden, ob es sich um ein gesetztes oder gelöschtes

Bit handelt. Für die Programm- und Datenspeicherung sind zwei verschiedene „Töne" (Zustände) notwendig: Einer für den Zustand „0" und ein anderer für den Zustand „1". Zwei verschieden lange Rechteck-Impulse, wobei der Impuls für ein gesetztes Bit doppelt so lang ist wie für ein gelöschtes Bit, kennzeichnen die Zustände. Auch bei gering eingestellter Lautstärke können Sie beim Laden und Speichern auf Kassette diese Töne hören.

Eine Diskette ist eine runde Plastikscheibe, die mit einem magnetisierbaren Material beschichtet und deren Größe abhängig vom benutzten System ist. Die DDI-1 benutzt das vollkommen unübliche Format von 3 Zoll. Warum sich die Firma Schneider (Armstrad) für dieses Format und nicht für die Formate 3 1/2 oder 5 1/4 Zoll entschieden hat, bleibt dahingestellt.
Der Vorteil einer Diskette gegenüber der Kassette liegt im wesentlichen darin begründet, daß auf eine Scheibe (Diskette) auch räumlich zugegriffen werden kann. Bei einem Band (Kassette) hingegen können Daten zwangsläufig nur hintereinander liegen.
Im Gegensatz zur Kassette muß die Diskette erst noch formatiert werden. Formatieren bedeutet, daß bestimmte Bereiche auf der Diskette für die Ablage von Daten vorbereitet und reserviert werden. Nach dem Formatieren befinden sich 40 Spuren (Tracks) mit je 9 Sektoren auf der Diskette. Auf jedem dieser 360 Blöcke (40*9=360) können 512 Bytes an Informationen abgelegt werden.

19.2 SAVE - Sichern von Programmen

Der SAVE-Befehl (save = sichern) gliedert sich in 4 verschiedene Funktionen:

- Sichern als normales Programm.
- Sichern als geschütztes Programm (P).
- Sichern als ASCII-Datei (A).
- Sichern als binäre Datei (B).

Ein im Speicher des CPC-464 vorhandenes Programm wird mit dem Befehl

SAVE "filename"

auf die Kassette/Diskette abgelegt. Beim Arbeiten mit dem Datacorder darf der „filename" 16 Zeichen beinhalten. Arbeiten Sie mit einer angeschlossenen Diskettenstation oder dem CPC-664, sind Filenamen nur bis zu einer Länge

von 8 Zeichen zulässig. Dabei ist es möglich (und sinnvoll), eine weitere Information über den Filetyp mit in den Filenamen einzubinden. Aus dieser Information kann z.B. hervorgehen, ob es sich um ein BASIC-Programm oder eine Textdatei handelt. Der Filetyp besteht aus bis zu drei Zeichen, die durch einen Punkt getrennt dem Filenamen nachgestellt sind. Hier einige Beispiele:

SAVE "TEST.BAS"	BASIC-Programm
SAVE "WERTE.DAT"	Daten-File
SAVE "ROUTINE.ASM"	Assembler Quellprogramm

Obwohl diese Kennzeichnung bei einer Kassettenaufzeichnung nicht notwendig ist, sollten Sie auch hier dieses Standardformat benutzen.

Jedes Programm läßt sich vor dem Kopieren schützen. Dazu folgt nach dem Filenamen ein durch Komma getrenntes „P" (P = protect = geschützt).

```
SAVE "GEHEIM.BAS",P
```

Ein so geschütztes Programm kann weder mit dem normalen LOAD-Befehl geladen noch gelistet werden (siehe 19.5).

Das Speichern eines Programms findet immer in einer verschlüsselten (verkürzten) Form statt. Alle BASIC-Befehle sind z.B. in sogenannten Token (siehe Band 2) verschlüsselt, die jeweils maximal 2 Bytes an Speicherraum benötigen. Durch das Speichern mit dem Befehl

```
SAVE "TEXT.ASC",A
```

wird das Programm als ASCII-Datei (A = ASCII) abgegelegt, d.h. so wie es beim Listen auf dem Bildschirm erscheint. Ein als ASCII-Datei abgelegtes Programm können Sie z.B. ohne weiteres in ein Textverarbeitungssystem einlesen.

Als letzte, aber auch interessanteste Möglichkeit bietet der CPC-464 die Form des binären Speicherns an. Unter binärem Speichern versteht man das Ablegen eines bestimmten Speicherbereiches auf Kassette/Diskette. Dabei speichert die Kassette/Diskette die Informationen (Bytes) so, wie sie vorgefunden werden.

```
SAVE "FILENAME.BIN",B,startadresse,länge
```

Nach dem Filenamen folgt ein „B" zur Kennzeichnung einer binären Datei. Die Startadresse bestimmt den Beginn des zu sichernden Speicherbereichs. Die Länge bestimmt die Anzahl der Bytes, die von der Startadresse an gespeichert werden sollen. Mit der binären Speicherung lassen sich Bildschirminhalte, Maschinenprogramme, Variableninhalte etc. nach Belieben sichern. So zum Beispiel der Bildschirmspeicher:

```
SAVE ''BILD.BIN'',B,49152,16*1024
```

19.3 LOAD - Laden von Programmen

Ein auf Kassette/Diskette befindliches Programm ist mit dem Befehl

```
LOAD ''FILENAME.TYP''
```

in den Speicher des CPC-464 zu laden (load = laden). Dabei wird ein im Speicher befindliches Programm zerstört (überschrieben). Handelt es sich bei dem zu ladenden File um eine binäre Datei, wird sie automatisch wieder an die Stelle geladen, von wo sie beim SAVE-Befehl geholt wurde. Durch Angabe einer Lade-Startadresse ist es weiterhin möglich, eine binäre Datei an eine beliebige Adresse zu verschieben.

```
(MEMORY 15999)
LOAD ''BILD.BIN'',16000
```

Die binäre Datei 'BILD' liegt nach dem Laden ab der Adresse 16000.

19.4 MERGE - Programme zuladen

MERGE (merge = zusammenführen) ermöglicht es, mehrere Programme in den Arbeitsspeicher des CPC-464 zu laden. Dabei überschreibt das nachgeladene Programm alle gleichnummerierten Programmzeilen. Aus diesem Grunde ist beim MERGEn darauf zu achten, daß sich die Programme (Zeilennummern) nicht überschneiden oder überlappen. Leider ist auf Grund eines Fehlers

im DOS (Disk Operating System) beim Diskettenbetrieb nur das Zuladen von ASCII-Daten möglich. Im Kassettenbetrieb hingegen lassen sich alle Programme zuladen.

```
LOAD "PROG1.BAS"
......
MERGE "PROG2.BAS"
```

Beim Zuladen von der Diskette muß das nachzuladende Programm vorher als ASCII-Datei (SAVE "FILENAME.ASC",A siehe 19.2) auf die Diskette abgelegt werden. Dann läßt sich dieses Programm als ASCII-Datei zu dem bestehenden Programm zuladen.

```
LOAD "PROG1.BAS"
......
MERGE "PROG2.ASC"
```

19.5 RUN "..." - Laden und Starten

Mit dem Befehl

```
RUN "FILENAME.TYP"
```

wird das bezeichnete Programm in den Arbeitsspeicher des CPC-464 geladen und anschließend gestartet. Besonders wichtig ist dieser Befehl zum Laden und Starten von geschützten Programmen (SAVE "FILENAME.BAS",P siehe 19.2). Kopiergeschützte Programme lassen sich ausschließlich in dieser Form laden und starten!!

19.6 CAT - Inhaltsverzeichnis

Nach dem Befehl

```
CAT
```

erscheint eine Liste der auf der Kassette/Diskette vorhandenen Files (catalogue = Verzeichnis). Beim Diskettenbetrieb erscheint nach kurzer Lesezeit sofort

die gesamte Liste der vorhandenen Files mit Angaben zum Programmtyp, sowie zur Länge des Files in kByte (1024 Bytes). Beim Kassettenbetrieb läuft das Band von der momentanen Stellung bis zum Ende der Kassette durch und gibt alle gefundenen Programmblöcke (a 2 kByte) wieder. Dabei wird neben dem Filenamen die Blocknummer und eine Kennzeichnung des Filetypes ausgegeben. Gängige Filtetypkennzeichen sind:

$ BASIC-Programm
% Geschütztes Programm
* ASCII-Datei
& Binäre Datei

Wurde der gelesene Block korrekt gefunden (es liegen keine Fehler vor), bestätigt dies der Rechner mit einer 'OK'-Meldung hinter dem Filetyp. Andernfalls folgt eine Fehlermeldung (Read error).

20.0 Dezimalsystem

Der Mensch benutzt üblicherweise das ihm bekannte Dezimal- oder Zehnersystem. Alle denkbaren Zahlen lassen sich mit 10 Symbolen, den Zeichen „0" bis „9" darstellen. Ist eine Zahl größer als 9, findet ein Übertrag in die nächsthöhere Potenz statt. Die Dezimalzahl 2345 läßt sich auch anders ausdrücken:

$$2345 = 2*1000 + 3*100 + 4*10 + 5*1$$

Mathematisch folgt daraus:

$$2345 = 2*10\hat{}3 + 3*10\hat{}2 + 4*10\hat{}1 + 5*10\hat{}0$$

Zahlen mit einer Null potenziert, ergeben immer 1 (z.B. 10^0=1, 3424^0=1 etc.). Eine Ausnahme bildet hierbei der Wert 0^0, denn er ist mathematisch nicht definiert. Somit ist jede endliche Zahl zu bestimmen.

Der Rechner ist nicht in der Lage, „bis Zehn zu zählen", denn er müßte sich dann 10 verschiedene Zustände merken können (von 0 bis 9). Er ist nur in der Lage, zwei Zustände zu unterscheiden: Strom fließt, Strom fließt nicht oder besser, ein Speicher ist geladen ('1') oder eben nicht ('0').

20.1 Binärsystem

Der Computer besitzt eine bestimmte Anzahl von Speicherstellen. Jede dieser Speicherstellen kann entweder geladen oder nicht geladen sein. Eine geladene Speicherstelle wird allgemein mit „1" und eine nicht geladene mit „0" bezeichnet. Diese Speicherstelle heißt „Bit" (eng.: binary digit). 8 Bits sind zu einem „Byte" (Kunstwort) zusammengefaßt und bilden für die Speicherung eine feste Einheit.

8 Bit = 1 Byte

Eine größe Anzahl von Bytes wird in „kByte" (nicht Kilo-Byte sondern „kByte') angegeben. 1 kByte entspricht 1024 Byte.

1 kByte = 1024 Byte = 8192 Bit

Der CPC-464 besitzt Speicherraum für 64 kByte, das sind:

64 kByte = 64*1024 = 65536 Bytes
oder 65536*8 = 524288 Bits

Um Werte größer als „1" darzustellen, benötig der Rechner mehr als ein Bit, denn ein Bit unterscheidet nur den Zustand „0" und „1".

00 = 0
01 = 1
10 = 2
11 = 3

Die beiden Bits sind jeweils an- oder ausgeschaltet. Durch die Kombination zweier Bits ist es möglich, 4 verschiedene Werte darzustellen. Durch das Kombinieren vieler Bits lassen sich wieder endliche Zahlen bilden.

10101 = 21

Der Übertrag in die nächsthöhere Potenz findet immer von 1 auf 2 statt (da die Basis 2 ist).

10101 = 1*16 + 0*8 + 1*4 + 0*2+ 1*1 = 21
10101 = 1*2^4 + 0*2^3 + 1*2^2 + 0*2^1 + 1*2^0

Mit einem Byte (8 Bits) sind insgesamt 256 verschiedene Kombinationen möglich und Sie können so die Zahlen 0 - 255 darstellen.

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0		
128	064	032	016	008	004	002	001		
0	0	0	0	0	0	0	0	=	0
0	0	0	0	0	0	0	1	=	1
0	0	0	0	0	0	1	0	=	2
0	0	0	0	0	0	1	1	=	3
0	0	0	0	0	1	0	0	=	4
0	0	0	0	0	1	0	1	=	5
0	0	0	0	0	1	1	0	=	6
0	0	0	0	0	1	1	1	=	7

0	0	0	0	1	0	0	0	=	8
. . . . .									
1	1	1	1	1	1	0	0	=	252
1	1	1	1	1	1	0	1	=	253
1	1	1	1	1	1	1	0	=	254
1	1	1	1	1	1	1	1	=	255

Die Umwandlung von Dezimal- in Binärwerte ist durch eine ständige Teilung durch 2 zu erzielen.

Umrechnung der Dezimalzahl 200 in Binär:

200 / 2 =	100	Rest	0
100 / 2 =	50	Rest	0
50 / 2 =	25	Rest	0
25 / 2 =	12	Rest	1
12 / 2 =	6	Rest	0
6 / 2 =	3	Rest	0
3 / 2 =	1	Rest	1
1 / 2 =	0	Rest	1

Die Binärzahl entsteht durch Lesen des Restes von unten nach oben.
200 = 11001000.

Für die Umrechnung einer dezimalen oder hexadezimalen Zahl in eine binäre Zahl steht im BASIC des CPC-464 ein kompletter Befehl zur Verfügung:

```
BIN$(wert,stellen)
```

„wert" darf eine Größe zwischen 65535 und -32768 annehmen, anderenfalls erscheint die Fehlermeldung „Overflow". Ist der Wert eine Fließkommazahl (keine ganze Zahl), wird der Wert automatisch gerundet. Ist die Anzahl der benötigten Zeichen (0 und 1) kleiner als die durch „stellen" angegebene Länge, wird der Text mit führenden Nullen versehen, bis die durch „stellen" gewünschte Länge erzielt ist.

```
PRINT BIN$(5,8)
00001001
```

Der Wert darf auch eine hexadezimale Zahl sein, dann muß er allerdings mit einem vorgestellten „&"-Zeichen gekennzeichnet werden:

```
PRINT BIN$(&FF,16)
0000000011111111
```

Der gesamte Speicherbereich des CPC-464 ist generell in Bytes (8 Bits) organisiert. Wird ein Speicher ausgelesen (PEEKen), erscheint immer die Dezimalzahl des Inhalts eines Bytes. Beim Beschreiben (POKEn) eines Speicherplatzes wird immer ein Byte gesetzt. Durch die 8-Bit-Struktur einer Speicherstelle kann nur ein ganzzahliger Wert zwischen 0 und 255 gePOKEt werden. Oftmals ist es allerdings notwendig, ein bestimmtes Bit eines Bytes zu verändern und alle anderen Bits unverändert zulassen.

20.2 Logische Arithmetik

Die beiden Operatoren „AND" (UND) und „OR" (ODER) sind Ihnen sicherlich von BASIC-Programmen her bekannt (Bereichsabfragen z.B. IF A=0 OR A=1 THEN...). Der AND- und OR-Operator ist ein logischer Verknüpfungsbefehl, mit dem auch zwei Zahlen binär miteinander verknüpft werden können.

1. AND: Die AND-Funktion verknüpft Binärzahlen nach folgendem Prinzip:

```
      10
AND   11
------
      10
```

Die beiden Zahlen „10" (Dez. =2) und „11" (Dez. =3) wurden „AND-Verknüpft". Die wertgleichen Stellen der beiden Zahlen werden verknüpft. Immer dann, wenn beide Stellen den Wert „1" tragen, ist das Ergebnis ebenfalls „1" ('1" UND „1" = „1"). Ist eine der Stellen „0", ist das Ergebnis auch Null.

„0" AND „0" = „0"
„0" AND „1" = „0"
„1" AND „0" = „0"
„1" AND „1" = „1"

Auch größere Zahlen (8 Bits - 1 Byte), lassen sich so verknüpfen. Hierzu ein Beispiel:

```
      10010111     = Dez. 151
AND   01011110     = Dez.  94
      ---------
      00010110     = Dez.  22
```

In BASIC ist dies der Eingabe

PRINT 151 AND 94

entsprechend. Probieren Sie die AND-Verknüpfung aus, damit Sie etwas „Gefühl" für Verknüpfungen erhalten.

2. OR: die OR-Verknüpfung ist etwas einfacher zu verstehen. Immer dann, wenn eine der beiden Stellen den Wert „1" besitzt, ist das Ergebnis ebenfalls „1".

„0" OR „0" = „0"
„0" OR „1" = „1"
„1" OR „0" = „1"
„1" OR „1" = „1"

Größere Zahlen verknüpfen sich entsprechend.

```
      10010111     = Dez. 151
OR    01011110     = Dez.  94
      ---------
      11011111     = Dez. 223
```

Im BASIC sieht es dann so aus:

PRINT 151 OR 94

Im CPC-464 steht noch eine weitere Verknüpfunsart, daß sogenannte „Exklusiv ODER" (XOR) zur Verfügung, welche der Vollständigkeit halber an dieser Stelle erklärt werden soll.

3. XOR Das Ergebnis einer XOR-Verknüpfung ist immer dann „1", wenn beide wertgleichen Bits ungleich sind:

„0“ XOR „0“ = „0“
„0“ XOR „1“ = „1“
„1“ XOR „0“ = „1“
„1“ XOR „1“ = „0“

Die XOR-Verknüpfung bietet sich vor allem zur Invertierung eines Bytes an (z.B. Bildschirmspeicher invertieren). Ein gesetztes Bit wird gelöscht, ein gelöschtes gesetzt:

```
    00100100     Das zu invertierende Byte
XOR 11111111     mit 255 XOR verknüpfen.
-----------
    11011011
```

Zum Beispiel:

```
POKE 49152, PEEK(49152) XOR 255
```

Mit den AND-, OR- und XOR-Befehlen ist es möglich, einzelne Bits eines Bytes zu manipulieren. Das Setzen eines bestimmten Bits ist recht einfach. Als Beispiel soll das Bit 3 eines Bytes gesetzt werden. Die Bits eines Bytes sind von rechts nach links von 0 bis 7 durchnumeriert.

```
Bit  76543210
-----------
     10010111
OR   00001000
-----------
     10011111
```

Das Bit 3 ist jetzt gesetzt und die restlichen Bits blieben unverändert, da sie mit „0“ OR-Verknüpft werden. In BASIC wird mit der Wertigkeit des zu setzenden Bits „OR-Verknüpft“.

```
PRINT 151 OR 8      oder      PRINT 151 OR 2^3
```

Allgemein ist das Setzen eines Bits mit der Eingabe

```
Wert OR 2^Bit      möglich.
```

Das Setzen wird vor allem dazu benutzt, um in Speicherstellen irgendetwas einzuschalten. Das bedeutet, daß der geänderte Inhalt sofort wieder in die Spei-

cherstelle zurückgeschrieben wird. Somit ist das Setzen eines Bits in eine Speicherstelle mit der Zeile

```
POKE Adresse, PEEK(Adresse) OR 2^Bit
```

zu erreichen. Anzumerken ist noch, daß ein bereits gesetztes Bit gesetzt bleibt.

Das Löschen eines Bits ist etwas verwirrender. Mit einer OR-Verknüpfung ist dieses nicht möglich, denn sowie ein Bit '1' ist, wird das Ergebnis ebenfalls '1'. Nur die AND-Verknüpfung hilft hier weiter. Das zu löschende Bit muß auf '0' gesetzt und alle anderen Bits auf '1' gelegt werden.

```
     10001011
AND  11110111
     ---------
     10000011
```

Das Bit 3 ist gelöscht. Überall dort, wo mit einer '1' AND-Verknüpft wird, bleibt der ursprüngliche Wert erhalten. So ergibt 150 AND 255 wieder 150! Der Wert des zu löschenden Bits ist also von 255 abzuziehen.

```
255 - 2^Bit
```

Ein Bit einer Speicherstelle ist mit

```
POKE Adresse, PEEK(Adresse) AND 255-2^Bit
```

zu löschen. Das Setzen und Löschen von bestimmten Bits ist speziell in der Grafik ein alltägliches Problem. Sie werden immer wieder vor der Aufgabe stehen, ein bestimmtes Bit eines Bytes zu setzen oder zu löschen.

20.3 Zerlegen in High- und Lowbyte

In einem Byte (8 Bit) lassen sich nur Werte bis 255 darstellen. Größere Werte müssen in zwei Bytes zerlegt werden. Damit der Computer alle seine 65536 Speicherstellen ansprechen und bestimmen kann (z.B. BASIC-Anfang, BASIC-Ende...), muß er zumindest Zahlen bis 65535 darstellen können. Alle

Stellen eines Bytes sind beim Wert 255 mit '1' gefüllt. Durch hinzuaddieren einer '1' müßte sich der Wert 256 ergeben. Dies würde allerdings einen Übertrag in die nächsthöhere Potenz erfordern, doch die ist nicht vorhanden.

```
      11111111     255
+     00000001       1
-----------------------
    1 00000000     256
```

Der Übertrag ist nicht mehr in dem Byte darstellbar. Das Byte selber hat dann den Inhalt Null! Nur ein zweites Byte kann die Aufgabe übernehmen, diesen Übertrag zu speichern. Dieses Byte wird als „Highbyte" (hochwertiges Byte) bezeichnet, der niederwertige Teil als „Lowbyte". Das Bit 0 des Highbytes entspricht nun nicht mehr dem Wert Dez. 1, sondern Dez. 256.

00000001 00000000 = 256

Daraus ist zu folgern, daß der Inhalt des Highbytes immer mit dem Faktor 256 zu multiplizieren ist.

Highbyte = 1, Lowbyte =0 = 1*256 + 0 = 256

Insgesamt lassen sich mit High- und Lowbyte

255*256 + 255 = 65535

Werte unterscheiden und darstellen.
Der Mikroprozessor ist unter bestimmten Bedingungen in der Lage, 2-Byte-Zahlen zu verarbeiten und zwar immer dann, wenn er auf eine seiner 65536 Speicherstellen zugreifen will. Die beiden Bytes liegen immer direkt hintereinander in der Reihenfolge Lowbyte, Highbyte. Mathematisch ist die Umrechnung recht einfach:

Highbyte = INT(Wert/256)
Lowbyte = Wert - Highbyte*256

Umgekehrt ist der Dezimalwert mit

Dezimal = Highbyte*256 + Lowbyte
32768 = 128*256 + 0

zu bestimmen.

20.4 Hexadezimalsystem

Noch vor Jahren war das Programmieren von Rechnern sehr umständlich: Speicherstellen mußten binär, also mit unendlich vielen Nullen und Einsen eingegeben werden. Als erstes Hilfsmittel für die Programmierung wurde das „Hexadezimalsystem" verwendet. Der große Vorteil lag (liegt) darin, daß jede Zahl von 0 bis 255 mit nur zwei Zeichen unterschieden werden kann. Heute ist die hexadezimale Darstellung zwar nicht mehr notwendig, hat sich aber derart „eingebürgert", daß sie immer noch verbreitet ist. Für Speicherauszüge wird die hexadezimale Darstellung immer noch bevorzugt, da alle Werte leichter zu überblicken sind (HEX-DUMP).
Das hexadezimale System beruht auf der Zahlendarstellung zur Basis 16, daher findet der Übertrag in eine höhere Potenz erst bei 16 statt. Es müssen also 16 Symbole für die Darstellung vorhanden sein. Für die Werte von 0 bis 9 können die normalen Ziffern benutzt werden. Die 10 ist allerdings nicht mehr mit einer Ziffer darstellbar, so daß hierfür ein Buchstabe „einspringen" muß.

Dez.	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Hex	&0	&1	&2	&3	&4	&5	&6	&7	&8	&9	&A	&B	&C	&D	&E	&F

Die hexadezimalen Zahlen werden zur Unterscheidung beim CPC-464 mit dem vorgestellten '&'-Zeichen gekennzeichnet. Vielfach wird jedoch auch ein führendes Hochkomma (') oder ein vor- oder nachgestelltes 'H' zur Kennzeichnung einer hexadezimalen Zahl benutzt.

&7 = 7 Dez
&A = 10 Dez
&F = 15 Dez

Der Übertrag findet bei 16 statt, somit ist

&10 = 1*16 + 0 = 16 Dez

Mit einer 2-stelligen Hex-Zahl lassen sich

&FF = 15*16 + 15 = 255

256 (0 - 255) dezimale Zahlen darstellen. Die Umrechnung einer beliebig großen Hex-Zahl (z.B. &A3FC) ist mit folgender Gleichung möglich

```
    A      3          F      C
10*16^3 + 3*16^2 + 15*16^1 12*16^0
```

Auch zur Umwandlung einer dezimalen oder binären Zahl besitzt der CPC-464 den komfortablen Befehl

```
HEX$(wert,stellen)
```

„wert" darf dabei zwischen 65535 und -32768 liegen. Wie beim BIN$-Befehl gibt „stellen" die Länge der auszugebenden Hex-Zahl (gegebenenfalls mit vorgestellten Nullen) an.

```
PRINT HEX$ (65535, 4)
FFFF
```

21.0 Memory - ein Spiel

Am Ende des ersten Bandes darf natürlich ein kleines Spiel nicht fehlen. Das Spiel „Super-Memory" ist dem bekannten Memoryspiel nachempfunden und ist für zwei Spieler(innen) ausgelegt. Ihre Aufgabe ist es, aus den 64 Karten die 32 Kartenpaare zu finden. Mit der Eingabe von Zeile und Spalte (durch Komma getrennt) können Sie jede beliebige Karte umdrehen. Haben Sie 2 gleiche Karten gefunden, dürfen Sie weiterspielen, andernfalls ist Ihr Gegner an der Reihe. Sind alle Karten aufgedeckt, endet das Spiel mit der Spielauswertung. Viel Spaß.

```
1000 REM ************************************* M.W.Thoma **
1010 REM *                                               *
1020 REM *              S U P E R - M E M O R Y          *
1030 REM *                                               *
1040 REM *************************************************
1050 MODE 1
1060 PRINT "      S U P E R  -  M E M O R Y"
1070 PRINT:PRINT
1080 PRINT"Versuchen Sie die beiden gleichen Karten"
1090 PRINT"zu finden. Insgesamt sind 32 Kartenpaare"
1100 PRINT"vorhanden. Fuer jedes gefundene Paar er-"
1110 PRINT"halten Sie einen Punkt und Sie duerfen  "
1120 PRINT"weiterspielen.Es ist nicht ganz einfach!"
1130 PRINT
1140 PRINT"Geben Sie bitte Ihre Namen ein ( bis zu ";
1150 PRINT"einer Laenge von 4 Zeichen !):"
1160 PRINT
1170 INPUT "Name Spieler 1 :";sp$(1)
1180 INPUT "Name Spieler 2 :";sp$(2)
1190 IF LEN(sp$(1)) >4 THEN sp$(1)=LEFT$(sp$(1),4)
1200 IF LEN(sp$(2)) >4 THEN sp$(2)=LEFT$(sp$(2),4)
1210 MODE 0
1220 WINDOW#0,1,20,21,25:REM  Eingabefenster
1230 WINDOW#1,1,20,1,20 :REM  Spielfenster
1240 '
1250 REM *************************************************
1260 REM *                                               *
1270 REM *              Aufbau des Spielfelds            *
1280 REM *                                               *
1290 REM *************************************************
1300 LOCATE#1,4,1
1310 FOR i=1 TO 8:PRINT#1, STR$(i);:NEXT i
1320 LOCATE#1,1,4
1330 FOR i=1 TO 8:PRINT#1, i:PRINT#1:NEXT i
1340 ORIGIN 112,104
1350 rf=2:zf=3
1360 FOR x=0 TO 7*64 STEP 64
1370 FOR y=0 TO 7*32 STEP 32
1380 GOSUB 2300
```

```
1390 NEXT y
1400 NEXT x
1410 PRINT "   Bitte warten ..."
1420 PRINT
1430 PRINT"    ich mische !"
1440 GOSUB 2470
1450 '
1460 REM ***************************************************
1470 REM *                                                 *
1480 REM *              Eingabe der Felder                 *
1490 REM *                                                 *
1500 REM ***************************************************
1510 n=n+1:IF n=3 THEN n=1
1520 CLS
1530 PRINT sp$(1);":";pu(1);TAB(10);sp$(2);":";pu(2)
1540 LOCATE 1,3:PEN 14:PRINT sp$(n);" ";
1550 PEN 1:INPUT "1. Karte :"; spa,zei
1560 IF zei<1 OR zei>8  THEN SOUND 1,100,10,3: GOTO 1540
1570 IF spa<1 OR spa>8  THEN SOUND 1,100,10,3: GOTO 1540
1580 IF sy$(zei,spa)="" THEN SOUND 1,100,10,3: GOTO 1540
1590 h1$=sy$(zei,spa):z=zei:s=spa
1600 x=(zei-1)*64
1610 y=(8-spa)*32
1620 zf=0:rf=14
1630 GOSUB 2300
1640 LOCATE#1,3+zei*2,2+spa*2:PRINT#1,sy$(zei,spa)
1650 LOCATE 1,4:PEN 14:PRINT sp$(n);" ";
1660 PEN 1;INPUT "2. Karte :"; spa,zei
1670 IF spa<1 OR spa>8 THEN SOUND 1,100,10,3: GOTO 1650
1680 IF zei<1 OR zei>8 THEN SOUND 1,100,10,3: GOTO 1650
1690 IF zei=z AND spa=s THEN SOUND 1,100,10,3: GOTO 1650
1700 IF sy$(zei,spa)="" THEN SOUND 1,100,10,3: GOTO 1650
1710 h2$=sy$(zei,spa)
1720 x=(zei-1)*64
1730 y=(8-spa)*32
1740 zf=0:rf=14
1750 GOSUB 2300
1760 LOCATE#1,3+zei*2,2+spa*2:PRINT#1,sy$(zei,spa)
1770 IF h1$=h2$ THEN 1960
1780 '
1790 REM ***************************************************
1800 REM *                                                 *
1810 REM *            Kein Kartenpaar gefunden             *
1820 REM *                                                 *
1830 REM ***************************************************
1840 FOR t=1 TO 500:NEXT
1850 SOUND 1,800,60,4
1860 rf=2
1870 x=(zei-1)*64:y=(8-spa)*32:zf=3:GOSUB 2300
1880 x=(z-1)*64:y=(8-s)*32:zf=3:GOSUB 2300
1890 GOTO 1510
1900 '
1910 REM ***************************************************
1920 REM *                                                 *
1930 REM *                Richtig gefunden  !              *
```

```
1940 REM *                                                    *
1950 REM ************************************************
1960 FOR i= 25 TO 350 STEP 25
1970 SOUND 1,i,20:SOUND 2,i*0.75,20:SOUND 4,i*1.5,20
1980 NEXT i
1990 pu(n)=pu(n)+1
2000 IF pu(1)+pu(2)=32 THEN  2130
2010 sy$(zei,spa)="":sy$(z,s)=""
2020 rf=2:ge=1
2030 x=(zei-1)*64:y=(8-spa)*32:zf=3:GOSUB 2300
2040 x=(z-1)*64:y=(8-s)*32:zf=3:GOSUB 2300
2050 ge=0
2060 GOTO 1520
2070 '
2080 REM ************************************************
2090 REM *                                                    *
2100 REM *                  Spielende erreicht !              *
2110 REM *                                                    *
2120 REM ************************************************
2130 MODE 1
2140 FOR i=50 TO 300 STEP 10
2150 SOUND 1,i,5:SOUND 2,i*0.75,6:SOUND 4,i*1.25,7
2160 NEXT i
2170 PRINT "        S U P E R - M E M O R Y"
2180 PRINT:PRINT
2190 IF pu(1)=pu(2) THEN PRINT."  Unentschieden":GOTO 2230
2200 PRINT " Gewonnen hat ";
2210 IF pu(1)>pu(2) THEN PRINT sp$(1):GOTO 2230
2220 PRINT sp$(2)
2230 PRINT:PRINT
2240 PRINT "Moechten Sie noch einmal spielen ?"
2250 a$=INKEY$:IF a$="" THEN 2250
2260 IF UPPER$(a$)="N" THEN MODE 1:END
2270 IF UPPER$(a$)="J" THEN RUN
2280 GOTO 2250
2290 '
2300 REM ************************************************
2310 REM *                                                    *
2320 REM *        Zeichnen der ausgefuellten Rechtecke        *
2330 REM *                                                    *
2340 REM ************************************************
2350 MOVE x,y
2360 DRAW x+64,y,rf
2370 DRAW x+64,y+32,rf
2380 DRAW x,y+32,rf
2390 DRAW x,y,rf
2400 IF ge=1 THEN 2450
2410 FOR i=2 TO 30 STEP 2
2420 MOVE x+4,y+i
2430 DRAW x+60,y+i,zf
2440 NEXT i
2450 RETURN
2460 '
2470 REM ************************************************
2480 REM *                                                    *
```

```
2490 REM *                Mischen der Karten                 *
2500 REM *                                                   *
2510 REM ***************************************************
2520 FOR i=1 TO 8
2530 FOR j=1 TO 8
2540 w=w+1:IF w=33 THEN w=1
2550 sy$(i,j)=CHR$(223+w)
2560 NEXT j
2570 NEXT i
2580 FOR i=1 TO 48
2590 z1=INT(RND(1)*8)+1
2600 z2=INT(RND(1)*8)+1
2610 z3=INT(RND(1)*8)+1
2620 z4=INT(RND(1)*8)+1
2630 sy$(0,0)=sy$(z1,z2)
2640 sy$(z1,z2)=sy$(z3,z4)
2650 sy$(z3,z4)=sy$(0,0)
2660 SOUND 1,100,1,7,1,1,1
2670 NEXT i
2680 SOUND 1,100,10,7,1,1
2690 RETURN
2700 REM ************************************************************
```

Register

8-Bit-Modus 111, 114
8080 13
Adressbus 13
AFTER 156
Akumulator 120
Analoguhr 159ff
AND 125, 176
Animation 35, 73
Anpassung 73ff, 96
ASCII 35, 111, 137
Assembler 115, 121
Auflösung 17, 60

Balkendiagramm 73, 92
Bank-Switching 14
BASIC-RAM 38
Betriebssystem 13
Bildschirmspeicher ... 25ff
Binärsystem 173ff
BIN$ 175
Bit 174
Blinken 21
BORDER 21
Bus 13
Busy 111
Byte 174

CAD 127
Cassette 167ff
Caps Lock 137
CAT 171
Centronics 111ff
CLG 93
CLS 20, 92
COMMOM 123
Control 137
COPY-Taste 119
Cursor 54, 63

Datenbus 13
Dauerfunktion 138
DEF FN 67
Dezimalsystem 173
DI 164
Directory 170
Diskette 167ff
DRAW 64
DRAWR 65
Drucker 111ff
EI 164
Ellipse 61
ELOCAD 127ff
ERL 152
ERR 152
ERROR 152
Erweiterungszeichen .. 139ff
EVERY 158ff

Farben 18ff
Fehlerbehandlung 97, 151ff
Fehlercode 151
Fenster 17, 28, 51ff, 91
Fensternummer 51ff
Formatieren 168
FRE 38
Funktionsplot 95ff
Funktionsübergabe ... 96
Funktionstasten 141ff

Gate Array 14
Gerätenummer 112
Grafik 57ff
Grafikhardcopy 113ff

Hardcopy 113ff
Hexdezimalsystem ... 181ff
HEX$ 182
Highbyte 179
HIMEM 38
Hires 17, 57ff

INK 20
Interpreter 14
Inverse 24
Interrupt 155ff

JOY 124
Joystick 124ff

Kassette 167ff
Kernal 14
KEY 140ff
KEY DEF 96, 138

Kleingrafik 42
Koordinatensystem ... 58ff, 91
Kreis 61
Kuchendiagramm 79

LOAD 170
LOCATE 54
Logische Operatoren .. 176ff
Lowbyte 179

MERGE 170
MODE 18, 25
MOVE 63
MOVER 64
Multicolor 30ff

Nibble 114
Nullpunkt 57, 95

ON ERROR GOTO .. 152
OR 177
ORIGIN 57, 91, 118

Paddle 123
PAPER 19
Parallelschnittstelle ... 111
PEN 19
Perspektive 68, 101
PLOT 59ff
PLOTR 63
POKE 43
Priorität 158

Quader 68ff, 84

RAM 13, 167
Read Error 172
Rechteck 65ff
REMAIN 162ff
Repeat 138
Restart 15
RESUME 153
RESUME NEXT 153
ROM 14
RUN „..“ 171

SAVE 168ff
Scrolling 52, 129
Shift 137
SPEED INK 22
Steuerzeichen 35, 145ff
Strobe 111
SYMBOL 41
SYMBOL AFTER ... 52, 129

TAG 82
TAGOFF 82
Tastatur 137ff
TEST 114
TESTR 114
Texthardcopy 119ff
TIME 155
Timer 22, 156ff
Tortendiagramm 79
Transparent 23, 83

Uhr 156
Unterbrechung 155ff
Ursprung 57

Werte-Plot 77
WIDTH 113
WINDOW 17, 28, 51ff, 91
WINDOW SWAP 55

XOR 146
XPOS 63

YPOS 63

Z80 13
Zeichengenerator 43ff
Zeichen-ROM 36
Zeichensatz 35ff

Mikrocomputer

Manfred Thoma

Brücke zum C-64

Ein erweitertes Handbuch

1985, 280 S., kart.,
DM 46,—
ISBN 3-7785-1095-9

Das vorliegende Buch bietet eine Zusammenfassung aller wesentlichen Programmiertechniken in den Bereichen

- Grafik
- Musik
- Diskette
- Cassette
- Joystick, Paddle, Lightpen, USER-Port etc.

Es ist ausschließlich in BASIC gehalten, so daß außer grundlegenden Kenntnissen der Programmiersprache BASIC keine weiteren Kenntnisse vorausgesetzt werden. C-64 Benutzer denen der Umgang mit binären und hexadezimalen Zahlen, sowie der logischen Arithmetik nicht geläufig ist, finden in den Anhängen grundlegende Erklärungen.

Das Buch kann als Arbeitsbuch im eigentlichen Sinne genutzt werden, d. h.: Man kann es Stück für Stück durcharbeiten, oder es aber wie ein Handbuch nutzen, da alle Abschnitte weitestgehend in sich geschlossen sind.

Dr. Alfred Hüthig Verlag
Im Weiher 10
6900 Heidelberg 1

Thomas Tai

Computer-Abenteuer

Mikrocomputer

Das Adventurebuch für den C-64

1985, ca. 120 S., kart.,
ca. DM 35,—
ISBN 3-7785-1132-7

Dieses Buch soll dem Adventurespieler helfen, ein System in die Lösung von Adventures, also den Computerabenteuer-Spielen zu bringen und diese somit um einiges zu erleichtern.
Zu diesem Zweck werden alle wichtigen Gesichtspunkte, die bei der Lösung von Adventures zu beachten sind aufgezählt und ausführlich erläutert. Dabei werden immer wieder aktuelle und bekannte Adventures als Beispiele erwähnt. Weiterhin wird auf typische Situationen und Schwierigkeiten eingegangen, vor denen der Spieler sehr oft steht. Die abgebildeten Entwurfsblätter, deren Gebrauch ebenfalls erklärt wird, erleichtern das Überwinden von Schwierigkeiten, machen die Probleme überschaubar und geben so die optimale Voraussetzung für eine vollständige Lösung des Adventures. Zur Veranschaulichung dienen die Lösungen von drei populären Grafikadventures.

Der letzte Teil des Buches befaßt sich mit der Bedienung und Ratschlägen für besonders komplexe und außergewöhnliche Adventures. Alle Programme, die in diesem Buch erwähnt werden, laufen auf dem C-64. Trotzdem ist dieses Buch für alle geeignet, die Spaß an der Lösung von Adventures haben; egal welchen Computer er besitzt.

Dr. Alfred Hüthig Verlag
Im Weiher 10
6900 Heidelberg 1

Mikrocomputer

Manfred Thoma

C 128-Praxis

Band 1

1985, ca. 160 S., kart.,
ca. DM 35,—
ISBN 3-7785-1148-3

Der Commodore 128 überzeugt durch sein überaus mächtiges BASIC V7.0. Die beachtliche Anzahl von Grafik-Befehlen werden anhand vieler, teilweise recht umfangreicher Programme erklärt. Dabei ist besonders darauf geachtet worden, daß alle Programme über das Demonstrationsniveau hinausgehen und praktisch anwendbar sind. Auch die Programmierung, des Commodore 64 bekannten und überaus erfolgreichen SID-Chips findet ihren Platz.
Neben der grundlegenden Einführung in den Aufbau und der Arbeitsweise des Commodore 128, die mit Hilfe des integrierten Monitors vorgenommen wird, finden Sie die Programmierung und Anwendung der neuen Diskettenstation VC-1571. Durch den schnellen Datenzugriff sind semiprofessionelle Programme entwickelt worden, die durchaus in einem kommerziellen Rahmen passen (Bücherdatei, Kontoführung etc.).

Dr. Alfred Hüthig Verlag
Im Weiher 10
6900 Heidelberg 1

Hajo Lemcke

Assembler mit dem C-64 (C-128)

Mikrocomputer

Band 1: Einführung

1985, ca. 160 S., kart.,
ca. DM 35,—
ISBN 3-7785-1091-6

Band 2: Programm-Entwicklungen

1985, ca. 150 S., kart.,
ca. DM 35,—
ISBN 3-7785-1153-X

Der erste Band gliedert sich in drei Hauptteile. Am Anfang steht eine Einführung mit Begriffs- und Aufbauerklärung. Außerdem ist ein Basic-Listing enthalten, mit dem ein voll funktionsfähiger Minimonitor zur Eingabe der folgenden Programme erzeugt wird.
Der zweite Teil des Buches besteht aus der Einführung in sämtliche Assemblerbefehle. Dabei wird die Sprachähnlichkeit genutzt, so daß die meistgebräuchlichen Befehle auch zuerst erklärt und eingesetzt werden. Zusätzlich sind die Befehle in logische Gruppen gegliedert. Zu jedem Befehle, bzw. zu jeder Befehlsgruppe gibt es ein lauffähiges Beispielprogramm, das sich später in große Programme einbauen läßt. Es folgen jeweils Fragen und Übungsaufgaben, deren Lösungen sich im Anhang befinden.

Der letzte Hauptteil enthält die für die Assemblerprogrammierung notwendigen Befehlstabellen; eine ASCII-Tabelle und sämtliche Adressierungsarten sind zusammengefaßt.
Im zweiten Band wird gezeigt wie die im ersten Band gewonnenen Kenntnisse in großen seiner professionellen Programmen zur Anwendung kommen.

Dr. Alfred Hüthig Verlag
Im Weiher 10
6900 Heidelberg 1

Der erste Band des zweibändigen Werkes beschäftigt sich schwerpunktmäßig mit den Grafikmöglichkeiten des Schneider CPC-464/664. Es ist besonders darauf geachtet worden, daß alle behandelten Probleme und Programmiertechniken einen konkreten Bezug haben. Alle Programme sind aus einem konkreten Bereich entwickelt und mit den behandelten Befehlen realisiert worden.

Neben unzähligen Beispielen findet der Benutzer viele Programme und Unterprogramme, die seine Arbeit mit dem Schneider CPC-464/664 erheblich erleichtert. Gezeigt werden Programme von der Multicolorgrafik bis hin zum kompletten CAD-System, vom Tortendiagramm bis zum 3D-Funktionspilot, von der 3-dimensionalen Wertdarstellung bis zur Analoguhr. Alle Programme sind so ausgelegt, daß sie ohne große Schwierigkeiten auch speziellen Wünschen angepaßt werden können. Trotzdem handelt es sich bei diesem Buch nicht um eine Programmsammlung im eigentlichen Sinne. Es ist vielmehr ein Arbeits- und Handbuch, welches durch praktische Anwendungen (Programme) ergänzt ist, die den professionellen Charakter des Schneider CPC-464/664 deutlich unterstreichen.

Neben der Grafik ist weiterhin die Centronics-Schnittstelle mit mehreren Hardcopy-Programmen, der Joystick, die Tastatur und die Interruptsteuerung ausführlich behandelt. Die Kassetten- und Diskettenbedienung findet ebenso Platz wie alle Besonderheiten des BASIC-Interpreters.

Hüthig

Thoma **CPC-464/664-Praxis**

AMSTRAD CPC

MÉMOIRE ÉCRITE
MEMORY ENGRAVED
MEMORIA ESCRITA

https://acpc.me/